تعلم الأسمبلي 32Bit

محمد فاروق فايد

(الجزء الأول)

Email: Muhammad.Fayed@hotmail.com

Phone: 0115325678 / 0110501575

بسم الله الرحمن الرحيم

الحمد لله رب العالمين والصلاة والسلام علي اشرف الأنبياء والمرسلين سيدنا محمد أما بعد ..

كنت منذ عام 2006 أفكر في عمل كتاب شامل لتعلم الأسمبلي 32 بت موجه نحو حديثي العهد بهذه اللغة .. ومن يريد تعلمها ولكنه لم يستطع ذلك لأسباب عديدة علي رأسها .. افتقار المحتوي العربي لدروس مسلسلة ومبسطة إلي أقصي درجة .. ولم أفكر في القيام بذلك جديا سوي في نهاية 2008 .. وحينها كنت أضيف إليه كل بضعة أيام شيئا ما .. ولكنه توقف عدة مرات لأسباب عديدة .. أخرها لمدة سنة كاملة نظرا لقبولي ولله الحمد في الكورس التدريبي لمعهد ITI (تحية خاصة لـ Intake30 المنصورة Big ups) وكنت أفكر في نشره علي شبكة الانترنت للقاريء العربي قبل عام .. ولكن قلت لنفسي أنتظر قليلا حتي تجعله أشمل وتضيف إليه موضوعات شيقة أخري كنت أود الخوض فيها .. ولكن يبدو أنه كلما تأخر المرء ازداد كسلا عن ما سبق .. ولابد أن تأتي دائما دماء جديدة لتدفع العجلة كل حين وأخر ...

ولهذا أكتفي ها هنا نظرا لظروف العمل والوقت الغير كافي لدراسة التقنيات الجديدة التي تتحفنا بها الشركات كل يوم .. وأترك الأمر لمن يأتي ويكمل ... وسأكون سعيدا لو أنه تعلم شيئا ولو بسيطا من هذا الكتاب.

وقد قمت بتقسيم هذا الكتاب إلي جزئين كي يسهل تصفحه وأرفق بين يديك الآن الجزء الأول وسيليه الجزء الثاني تباعا.

فقط نقطة أخيرة .. وهي بخصوص الـ Credits .. فقد اعتمدت أثناء تعلمي لهذه اللغة علي دروس كلا من Iczelion و Thomas وأشخاص غيرهم قد نسيتهم للأسف .. وأخيرا مقالات كلا من JAAS وأحمد غريب في موقع الفريق العربي للبرمجة .. الأثنان لهم جزيل الشكر قد دفعوني دفعا لتعلمها دون أن يدروا .. والشكر العميق كذلك لأخي أحمد الذي علمني كتابة البرامج ووضعني علي هذا الطريق.. وفي النهاية أشكر وأحمد الله علي خروج هذا العمل إلي النور واسألكم خير الدعاء.

محمد فاروق
أغسطس - 2010



الفصل الأول

الأنظمة العددية

Numerical Systems

نستخدم نحن البشر النظام العددي العشري Decimal System في كل وقت.. وفي جميع تعاملاتنا اليومية , والنظام العشري به الأ ساس 10 .. أي أنه كما نعرف مكون من 10 أعداد (9 - 0).
ومنهم تتألف جميع الأعداد .. ونستطيع أن نقوم بالعمليات الحسابية علي تلك الأعداد مثل الجمع والطرح والضرب والقسمة .

ولننعش ذاكرتنا قليلا ولنري عدد عشري مثل 156 كيف يتكون في النظام العشري ذو الأساس 10 ..

	الآحاد	العشرات	المئات
الرقم	6	5	1
المعني	6×10^0	5×10^1	1×10^2
القيمة	6	50	100

حيث يتم حساب قيمة كل رقم بالصيغة { الرقم × (أساس النظام) مرفوع لأس موضعه في العدد } .. أي (الرقم × 10 مرفوع لأس الموضع)

وبجمع القيم فيظهر لنا العدد 156 مرة أخري ..

ولكن الحاسب لا يستطيع التعامل مع النظام العشري .. حيث لا يحتاج لتمثيل البيانات ونقلها وتخزينها سوي لعددين فقط ليمثلوا (وجود شحنة أم لا مفرغة).

وجود شحنة تمثل بالرقم واحد وعدم وجودها يمثل بالرقم صفر .. ومن العددين صفر وواحد يتم تكوين جميع الأرقام التي نريدها ..

وبالتالي فقد استخدم النظام الثنائي Binary System في كيان الحاسب .. والنظام الثنائي مكون من رقمين فقط (0 - 1) ..

وفي النظام العشري لمعرفة قيمة الرقم الحقيقية داخل العدد كنا نستخدم الصيغة السابقة : (الرقم × 10 مرفوع لأس الموضع)
وفي النظام الثنائي سنستخدم : (الرقم × 2 مرفوع لأس الموضع)

وكل نظام يمكن التحويل منه للأخر ..أي أنه يمكن التحويل من النظام العشري للثنائي .. وكذلك من النظام الثنائي للعشري .

ولنأخذ مثال للرقم الثنائي b(11001) ولا ينطق عشريا (أحد عشر ألفا وواحد) لأنه عدد ثنائي لا يتبع للصيغة العشرية .. بل ينطق (واحد واحد صفر صفر واحد) ..وسنقوم بتحويل هذا العدد الثنائي إلي النظام العشري هكذا :

الرقم	التحويل	القيمة
1	1×2^{0}	1
0	0×2^{1}	0
0	0×2^{2}	0
1	1×2^{3}	8
1	1×2^{4}	16

وبجمع القيم ينتج لنا الرقم العشري 25 .. ولتحويل الرقم العشري 25 إلي النظام الثنائي سنتبع الطريقة التالية :

الرقم	عملية القسمة	ناتج القسمة	باقي القسمة
25	25/2	12	1
12	12/2	6	0
6	6/2	3	0
3	3/2	1	1
1	1/2	0	1

بمعني أننا سنقسم الرقم 25 علي أساس النظام الثنائي وهو 2 .. وإذا كان هناك باقي من ناتج القسمة (مثل 3/2) فيكون واحد .. وإذا لم يتبقي شيء مثل (6/2) فيكون صفر ..

ثم نأخذ عمود باقي القسمة ونرتبه تنازليا من اليمين لليسار لينتج لنا العدد الثنائي b(11001) والمساوي للعدد العشري (25) ..

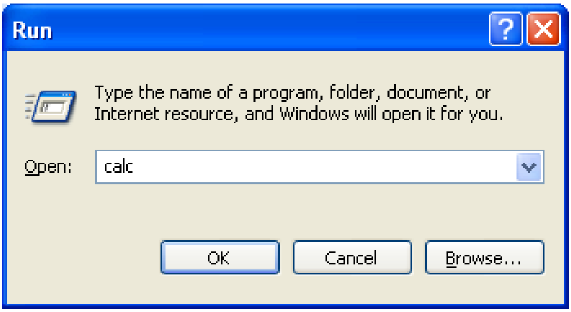
والجدول التالي يوضح بعض بدايات الأعداد العشرية والمقابل لها في الأعداد الثنائية :

العدد بالنظام العشري	العدد بالنظام الثنائي
0	0
1	1

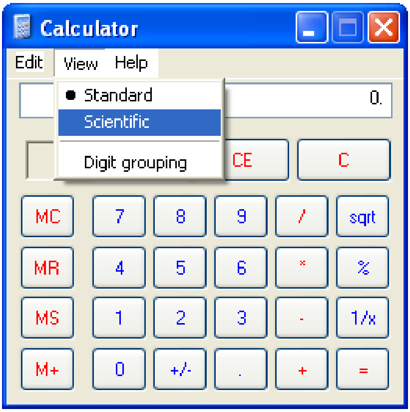
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15
10000	16

ولا داعي عندما تريد التحويل بين الأنظمة إلي استعمال الطريقة اليدوية .. فيكفيك فهمها .. ومن ثم يمكنك الاستعانة بالآلة الحاسبة التي تأتي مع الويندوز ..في عملية التحويل ..

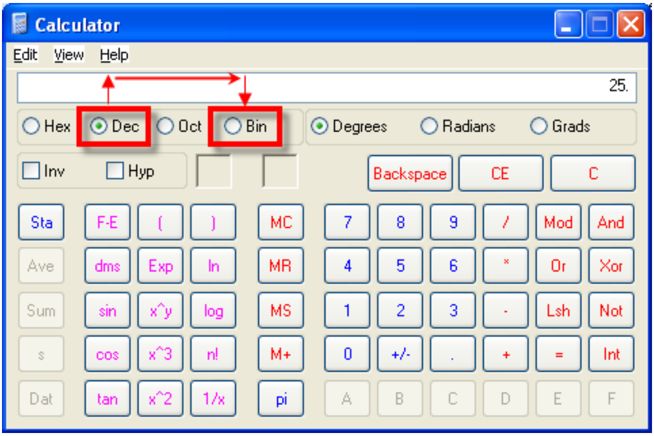
من قائمة Start اختر Run .. ثم اكتب Calc واضغط Ok.



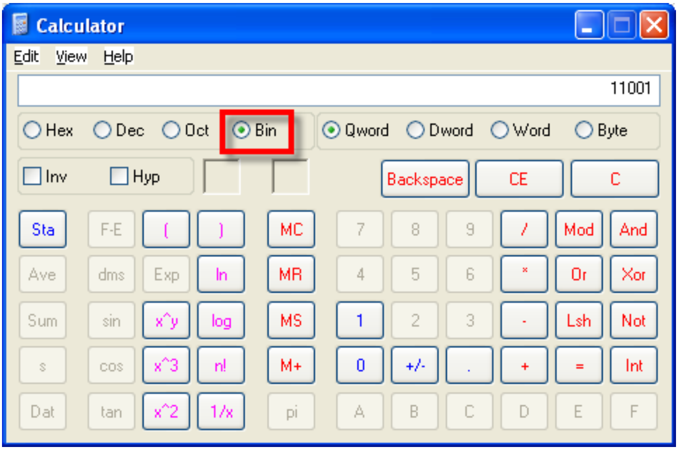
ثم نختار الوضع Scientific من قائمة View.



ثم نكتب الرقم الذي نريد تحويله من عشري لثنائي ومن ثم نضغط علي Bin .. كما هو موضح في الصورة التالية ..



وستتم عملية التحويل كما هو موضح في الصورة التالية ..



ويمكنك الآن تجربة تحويل المزيد من الأعداد العشرية والثنائية يدويا والتأكد من الناتج بسرعة باستخدام الآلة الحاسبة.

العمليات الحسابية علي الأعداد الثنائية

يقصد بالعمليات الحسابية المعروفة وهي الجمع والطرح والضرب والقسمة .. وسنأخذ ما يمكن أن يفيدنا وهما عمليتي الجمع والطرح .. وإذا أردت المزيد فيمكنك الرجوع إلي أي مصدر علي الويب ..

وعملتي الجمع والطرح .. تستطيع تطبيقها بسهولة علي النظام العشري فمثلا إذا قلت لك ناتج جمع (3+2) فستقول لي (5) ..
فنفس عملية الجمع للأرقام المكافئة لـ 2و3 في النظام الثنائي وهما (101) = (10+11)

وذلك تبعا لقاعدة الجمع التالية للنظام الثنائي:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

حيث (1+1) تساوي صفر ويتبقي معنا القيمة (1) لتجمع علي العمود التالي لعملية الجمع ..

ففي المثال السابق :

```
  1 0          2
+            +
  1 1          3
--------     --------
  101          5
```

مثال أخر :

```
  1110         14
+            +
  1011         11
---------    --------
 11001         25
```

أما بالنسبة لعملية الطرح فتخضع للقاعدة التالية :

0 – 0 = 0
1 – 1 = 0
1 – 0 = 1
0 – 1 = 1 (ونستعير واحد من أقرب عمود به القيمة واحد)

فمثلا إذا أردنا ناتج عملية الطرح التالية للأعداد الثنائية (101-10) والتي تكافئ في النظام العشري (2-3)

```
  11           3
 -            -
  10           2
--------     --------
  01           1
```

مثال أخر :

```
 11001        25
-            -
  1011        11
---------    --------
  1110        14
```

العمليات المنطقية علي الأعداد الثنائية

سنتناول بعض العمليات المنطقية مثل AND,OR,XOR, علي الأعداد الثنائية ..

أولا – المعامل AND :

ويستخدم تبعا للقاعدة التالية :

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

مثال :

```
        10                  2
And                   And
         11                3
      -------           -------
         10                 2
```

ويمكنك التحقق سريعا باستخدام الآلة الحاسبة وستجد بها جزء للمعاملات المنطقية مؤشر عليه في الصورة التالية



ثانيا – المعامل OR:

ويستخدم تبعا للقاعدة التالية :

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

مثال :

10	2
OR 11	OR 3
--------	--------
11	3

ثانيا – المعامل XOR:

ويستخدم تبعا للقاعدة التالية :

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

مثال :

```
      100                  4
XOR                  XOR
       110                6
    -------           -------
       010                 2
```

النظام السداسي عشر Hexadecimal

أساس هذا النظام هو 16 أي أنه يتكون من 16رقم .. يتقابل مع النظام العشري من 0-9 ثم يختلف من بعد ذلك ليصبح :

A = 10 : B = 11 : C = 12 : D = 13 : E = 14 : F = 15

Decimal	Binary	Hexadecimal
0	0000b	0h
1	0001b	1h
2	0010b	2h
3	0011b	3h
4	0100b	4h
5	0101b	5h
6	0110b	6h
7	0111b	7h
8	1000b	8h
9	1001b	9h
10	1010b	Ah
11	1011b	Bh
12	1100b	Ch
13	1101b	Dh
14	1110b	Eh
15	1111b	Fh

10h	10000b	16
11h	10001b	17
12h	10010b	18
13h	10011b	19
14h	10100b	20

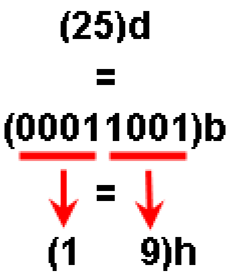
من مميزات النظام السداسي عشر تسهيل كتابة الأرقام الثنائية ..

فمثلا افترض أنك تريد كتابة (111111111111111111111111)b فتمثيلها بالنظام السداسي هو FFFFFF.. لذا فهو يسهل علينا قراءة وكتابة القيم لنا كبشر .. ولكن الحاسب لا يتعامل إلا مع النظام الثنائي 0,1 .

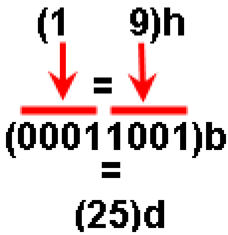
وسنتطرق الآن إلي التحويل من النظام السداسي عشر إلي العشري والعكس ..
وبالتالي نريد تحويل الرقم (19)h إلي النظام العشري ..

الرقم	التحويل	القيمة
9	9×16^{0}	9
1	1×16^{1}	16

وبجمع القيم ينتج لدينا العدد العشري(25) وإذا حولت هذا العدد إلي النظام الثنائي فسيكون (11001)b ..
وكل أربع أرقام في العدد الثنائي يمثلوا رقم بالنظام السداسي عشر ,
وبالتالي إذا قمت بتقسيم العدد (11001)b علي النحو الموضح بالصورة التالية.. فسيظهر لديك العدد بالنظام السداسي عشر.



ومن الممكن عكس الطريقة بالتأكيد ..للتحويل من النظام السداسي عشر للنظام العشري.. وقد درست فيما سبق كيفية التحويل من النظام الثنائي إلي النظام العشري ..



الآن نتطرق إلي التحويل من النظام العشري إلي النظام السداسي عشر بطريقة مباشرة بدون المرور بالنظام الثنائي كما في الطريقة السابقة ..

سنقوم في هذا المثال بتحويل القيمة (25)d إلي النظام السداسي .. وطريقة التحويل لن تختلف عن طريقة تحويل النظام العشري للثنائي والتي تطرقنا إليها من قبل .. الاختلاف فقط سيكون في القسمة علي أساس النظام السداسي عشر وهو16 ..

الرقم	عملية القسمة	ناتج القسمة	باقي القسمة
25	25/16	1	9
1	1/16	0	1

وبترتيب نواتج عمود باقي القسمة تنازليا من اليمين لليسار .. يظهر لدينا نتائج التحويل وهو العدد (19)h.



الفصل الثاني

أساسيات لغة التجميع

1. ما هي لغة التجميع.
2. كيف تبدأ مع لغة التجميع.
3. شفرات OPCODE.
4. المسجلات Registers.
5. القفزات الشرطية.
6. مميزات استخدام الأسمبلر MASM.

ما هي لغة التجميع Assembly ؟

تم بناء لغة التجميع لتكون بديلا لصف الأكواد الثنائي الذي يفهمه المعالج .. فمن الصعب جدا بالنسبة لنا نحن البشر كتابة برنامج ما بتعليمات المعالج في صورة صف بيانات ثنائي Binary .. ولذلك كان لابد من وجود لغة وسيطة تربط بيننا وبين المعالج ..

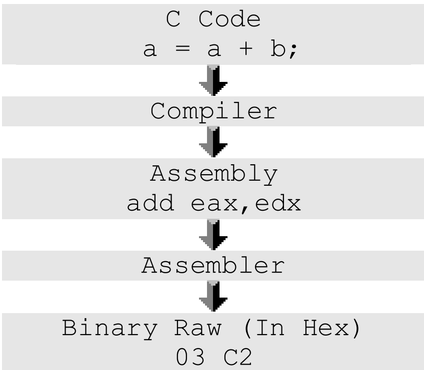
وبالتالي كل تعليمة في الأسمبلي تحول من قبل المعالج إلي بيانات ثنائية مثل:

```
add eax, edx
```

تعليمة add السابقة تستخدم لجمع القيمتين الموجودتين في كلا من المسجلين eax و edx وهذه المسجلات موجودة في ذاكرة المعالج ..

ويقوم المعالج بتحويل التعليمة السابقة إلي الصورة [03 C2] - وهي مكتوبة هنا بصيغة Hex- ومن ثم يقوم المعالج بتنفيذ التعليمة التي يمثلها ..

وأغلب لغات المستوى العالي مثل الـ C تحول عند ترجمتها إلي الأسمبلي ومن ثم يحولها الاسمبلر إلي صف البيانات الثنائي .. والشكل التالي يوضح فقط كيف تبدو تلك العملية.



كيف تبدأ مع لغة التجميع ؟

للبرمجة بلغة التجميع ستحتاج إلي الأدوات التالية :

☒ **المجمع Assembler**

وهو البرنامج المسئول عن تحويل شفرة الأسمبلي إلي صف البيانات الثنائي الذي يفهمه المعالج ويكون الملف الناتج من عملية التحويل object بامتداد (.obj).

وسنعتمد في هذا الكتاب علي مجمع Microsoft Macro - MASM Assembler .. وهناك عدة مجمعات أخري مثل NASM, TASM .. ويمكنك استخدامها بالطبع .. ولكن الموضوع سيكون أصعب قليلا في متابعة دروس الكتاب لاختلاف بعض صيغ التعليمات من مجمع لأخر.

ويمكن الحصول علي حزمة كاملة من MASM من الملفات الملحقة مع الكتاب

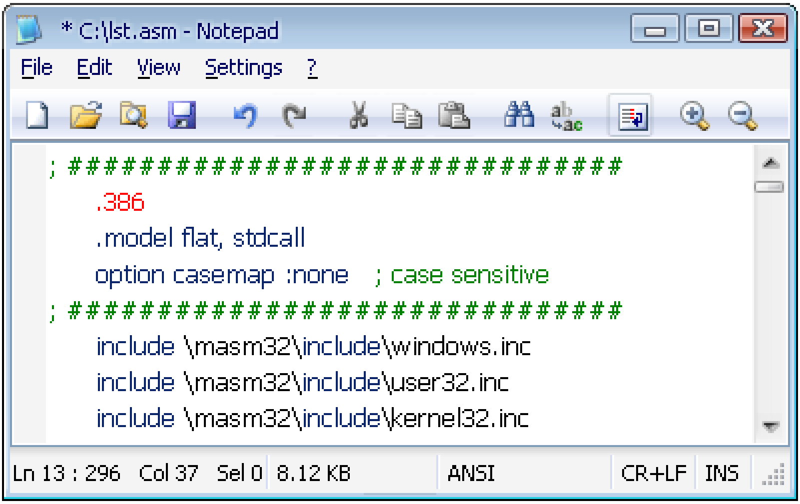
☒ **المربط Linker**

وهو البرنامج المسئول عن ربط جميع ملفات الـ Obj ومكتبات الـ DLLs وملفات المصادر Resource Files لإنتاج الملف التنفيذي النهائي EXE.

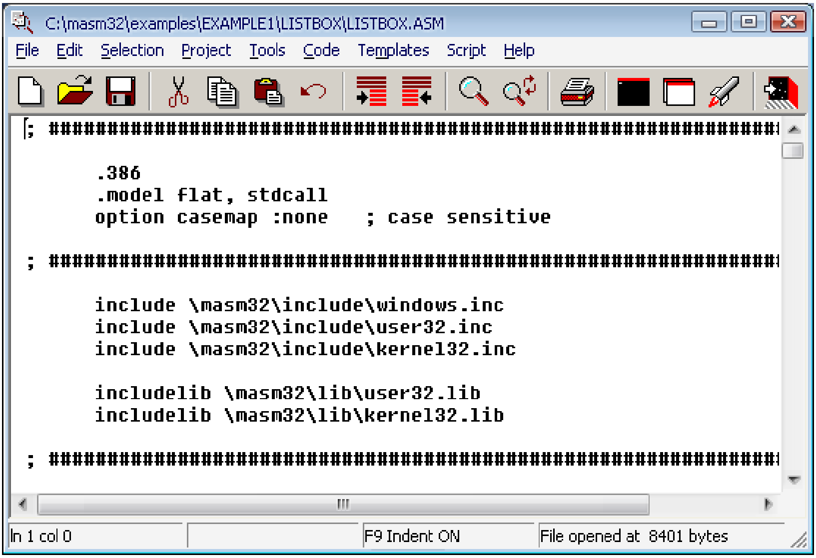
وستجد برنامج Link.exe مع حزمة MASM السابقة.

☒ **محرر شفرة نصوص Text Editor**

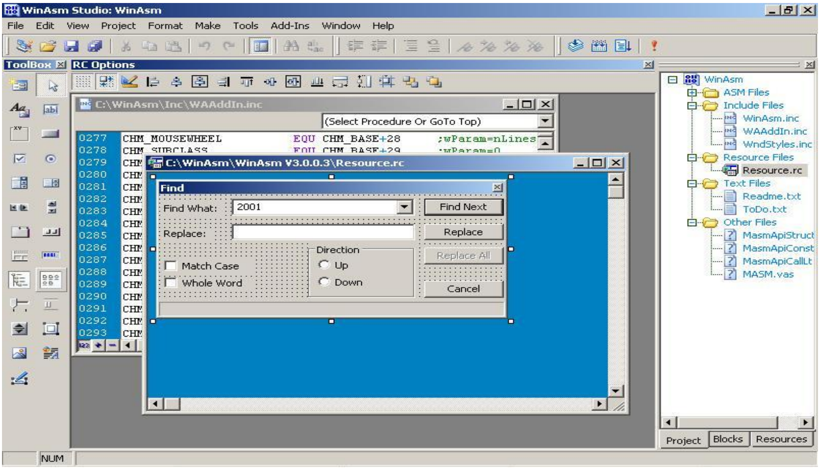
ويمكنك اختيار محرر النصوص الذي تفضل الكتابة به .. فيمكنك الاعتماد علي المحرر الشهير Notepad أو الترقية إلي المحرر المتميز Notepad2 أو Notepad++ واللذان يدعمان سكربت ملون للغة الأسمبلي ..
ويمكنك الحصول علي Notepad2 من الرابط :
http://www.flos-freeware.ch
و Notepad++ من الرابط :
http://notepad-plus.sourceforge.net/



ستجد أيضا مع حزمة MASM السابقة محرر الشفرة MASM32 Editor وهو محرر جيد ..



وسنعتمد الآن علي تطبيق Notepad2 لتحرير الشفرة .. ومن ثم سننتقل إلي بيئة تطوير مرئية متقدمة IDE مثل Visual WinAsm Studio أو RADASM تدعم التصميم المرئي لشكل البرنامج وخصائص عديدة كالـ intellisense مثل الموجودة في جميع بيئات اللغات المرئية .. فقط بعدما نتناول الأجزاء الرئيسية في بناء تطبيق يدويا يعمل تحت نظام ويندوز ..



مرجع لدوال الـAPI

سيوفر المرجع تفاصيل لدوال المكتبات الرئيسية مع شرح لها وكيفية استخدامها ويمكننا الاعتماد علي (win32.hlp) وهو مرجع لدوال أهم المكتبات مثل (kernel32, User32, GDI) والتي سنستخدمها في الكتاب.

وستجد هذا المرجع مع الملفات الملحقة بالكتاب.

إلي هنا معنا ما يلزمنا من أدوات تكفي للبداية .. ولكننا سنؤخر تلك البداية وكتابة أول تطبيق بهذه اللغة إلي الفصل القادم .. حتى نتعرف علي أساسيات هذه اللغة .. وهو ما سنتناوله في الجزء القادم.. ومهم جدا قراءة هذه الأساسيات قبل الدخول إلي كتابة تطبيقات Win32.

أساسيات لغة التجميع

1 - تعليمات OPCode

يتم كتابة شفرات الأسمبلي بواسطة تعليمات الـ OPCodes والتي تعتبر تعليمات يستطيع من خلالها المعالج فهم الأوامر المدخلة إليه ..والصيغة العامة لشفرات OPCodes تتكون من اسم التعليمة مع وجود معامل أو اثنين أو اسم التعليمة فقط بدون أيه معاملات.

فمثلا التعليمة Add تستخدم لجمع قيمتين بهذا الشكل :

```
add eax,edx
```

في التعليمة السابقة نلاحظ أن لها معاملين operands هما المصدر [Source] والهدف [Destination] , وتقوم تعليمة Add بإضافة قيمة المصدر [edx] إلي قيمة الهدف [eax] ..ويكون ناتج الإضافة في الهدف [eax].

وهناك تعليمات تحتوي علي معامل واحد فقط مثل تعليمة [inc] والتي تستخدم لزيادة قيمة المسجل بمقدار واحد , أو تعليمة [dec] وهي عكس التعليمة [inc] تستخدم لإنقاص قيمة المسجل بمقدار واحد.

```
inc eax

dec eax
```

وهناك تعليمات ليس لها أي معاملات مثل تعليمة Ret

```
ret
```

ومعاملات التعليمات من الممكن أن تكون نوعها إما مسجلات Registers , أو عناوين ذاكرة Memory Locations , أو قيم عددية ..

وسنعود إلي استكمال شفرات الـ OPCode بعد قليل .. فقط بعدما تتعرف علي ماهية المسجلات.

2 - المسجلات Registers

الـ Bit هو أصغر وحدة لتمثيل البيانات في الحاسب .. كل بت يستطيع حمل رقمين فقط (إما 0 أو 1) .. وبالتالي البت بمفرده لا يصلح استخدامه لتخزين البيانات لأنه لا يستطيع سوي حمل رقمين ..

بالتالي فيوجد وحدة أكبر من البت وهي الـ Nibble وهذه الوحدة تستطيع حمل 4 بت .

ولكن هذه الوحدة ليست بالحجم المطلوب لتخزين البيانات .. لأنه علي سبيل المثال إذا أردنا تخزين عدد مثل 25 .. فالمقابل الثنائي له (11001) أي (0001-1001) أي انه يحتاج إلي اثنين من وحدة الـ Nibble لتخزينه.

وبالتالي يلزم وجود وحدة أساسية يستطيع المعالج التعامل معها ويمكنها تمثيل جميع البيانات .. وهي البايت Byte .. والبايت يتكون من 8 بت .. وكل ما تقوم بفعله علي الحاسب سواء نقل بيانات أو تخزين أو استقبال بيانات من النت .. فالحاسب يتعامل معه في صورة بايتات ..

والبايت الواحد (8 بت) يستطيع حمل 2^8 عدد = 256 عدد .. أي يستطيع البايت حمل الأعداد غير المؤشرة *Not Signed* أي الموجبة من القيمة 0 إلي القيمة 255 .

وبمعرفة أن كل بت له 2 احتمال فقط .. بالتالي إذا أردت معرفة عدد الاحتمالات التي تستطيع وحده بحجم 8 بت حملها , فاستخدم العلاقة (عدد الاحتمالات مرفوعة لأس عدد الخانات)

ولنأخذ مثال مثلا العدد 25 يتم تخزينه في البايت الواحد المكون من 8 بت علي هذا النحو :

7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	1

High Order Nibble   *Low Order Nibble*

بعد ذلك تأتي لدينا وحدة الكلمة Word وتتكون من 2 بايت (16 بت) ..
ولنفترض مثلا أننا نريد تخزين عدد وليكن 10000 والمقابل الثنائي له (10011100010000)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0

High Order Byte　　　　Low Order Byte

ونستنتج كذلك أن وحدة الـ Word تستطيع حمل 2^{16} أي 65536 احتمال أي من القيمة 0 إلي القيمة 65535 وذلك للأعداد الغير مؤشرة ..

بعد ذلك تأتي لدينا وحدة الكلمة المضاعفة Double Word وتتكون من اثنتان من الـ Word أي 4 بايت (32 بت) .. وبالتالي تستطيع تلك الوحدة حمل 2^{32} احتمال.

بعد المقدمة السابقة نعود إلي المسجلات Registers .. ففي برامج 32 bit يتم استخدام مسجلات بحجم 32 بت , وبعض هذه المسجلات تكون جزء من مسجلات أخري .. فعلي سبيل المثال .. إذا كانت قيمة المسجل EAX هي 11223344h فهذا يعني الآتي :

```
EAX   =  11   22   33   44
AX    =  11   22   33   44
AH    =  11   22   33   44
AL    =  11   22   33   44
```

المسجلات AX,AH,AL هي أجزاء من المسجل EAX , المسجل AX هو مسجل بطول 16 بت وهو الكلمة السفلي Lower (الـ 2 بايت) من المسجل EAX , بينما المسجل AH البايت العلوي High (8 بت) من المسجل AX , وأخيرا المسجل AL هو البايت السفلي Low (8 بت) من المسجل AX.

```
AX  = 3344h      (16 bit)
AH  = 33h        (8  bit)
AL  = 44h        (8  bit)
```

EAX (32 bit)		
(16 bit)	AX (16 bit)	
(16 bit)	AH (8 bit)	AL (8 bit)

جميع معالجات عائلة x86 التي قامت شركة آنتل بإنشائها منذ عام 1978 تستخدم مسجلات بحجم 16 بت .. والاختلاف الملحوظ بين المعالجات كان في الأداء والسرعة وحجم الذاكرة التي يستطيع التعامل معها.

(1978) 8086
(1979) 8088
(1980) 80186
(1980) 80188
(1982) 80286

وبتطور المعالجات ظهر أول معالج وهو 80386 عام 1985 يحوي مسجلات بطول 32 بت والـ E تشير إلي Extended.. ويحتوي المعالج علي أكثر من نمط للأداء :

النمط الحقيقي Real Mode : وهو نمط يعمل كالمعالج 8086 لتشغيل برامج 16 بت التي كانت وما زالت تعمل علي معمارية 8086.

النمط المحمي Protected Mode : وو نمط يتيح تشغيل أكثر من برنامج في نفس الوقت حيث يكون نمط متعدد المهام Multi Tasking.

النمط 8086 العملي VIRTUAL 8086 MODE : وهو نمط لتشغيل أكثر من برنامج 8086 في نفس الوقت.

ثم تم تحسين المعالج 80386 وإنشاء معالج جديد وهو 80486 عام 1989 .. وتم إضافة ذاكرة ملحقة بالمعالج Cache memory .. وهي ذاكرة سريعة تستخدم لتخزين البيانات بين المعالج والذاكرة RAM .. وتم تحسين سرعه المعالج والتي وصلت لـ 100 ميجا هرتز.

ثم قامت شركة آنتل بإنتاج عائلة Pentium المستمرة حتي الآن بإصداراته المختلفة .. والمعتمدة علي نفس معمارية المعالج 80386 مع تحسين السرعة والأداء وزيادة الذاكرة الملحقة بالمعالج.

ومع عائلة Pentium تم استخدام مسجلات بطول 64 بت.. كمثال المسجل RAX يتكون من :

RAX (64 bit)			
(32 bit)	EAX		
	(16 bit)	AX	
		AH	AL

والـ R تشير إلي Really Extended.

تستطيع النظر إلي معمارية المعالج وتعليماته الأساسية من صفحة الـ MSDN التالية :

http://msdn.microsoft.com/en-us/library/cc267764.aspx

وفي الجزء القادم سنتعرف علي التركيب الداخلي لمعالجات عائلة x86 وسنتعرف أكثر علي المسجلات الموجودة بالمعالج وأنواعها واستخداماتها.

أنواع المسجلات Registers Types

1 - مسجلات **الأغراض العامة** General Purpose Registers
2 - مسجلات **المقاطع** Segment Registers
3 - مسجلات **العنونة** Adressing Registers
4 - مسجل **الرايات** Flags Register

1- مسجلات الأغراض العامة

وهي تستخدم لتخزين البيانات وإجراء العمليات الحسابية والمنطقية عليها .. وأيضا لتنفيذ أوامر مختلفة.

وهي مسجلات بطول 32 بت ومن الممكن استخدامهم لأي غرض .. بالرغم من أن لهم أسماء محددة وهم :

- eax (ax/ah/al) Accumulator Register

الاستخدام العام : يستخدم للعمليات الحسابية والإدخال والإخراج .. وأيضا لتخزين ناتج بعض استدعاءات المقاطعاتInterrupts.

- ebx (bx/bh/bl) Base Register

الاستخدام العام : يستخدم كمؤشر للوصول إلي الذاكرة .. وأيضا يستخدم لتخزين قيمة معادة من بعض الاستدعاءات لمقاطعات.

- ecx (cx/ch/cl) Counter Register

الاستخدام العام : يستخدم كعداد للتكرارات.

- edx (dx/dh/dl) Data Register

الاستخدام العام : يستخدم لعمليات حسابية (الضرب والقسمة) والإدخال والإخراج وبعض استدعاءات المقاطعات.

2- مسجلات المقاطع

وهي تستخدم لتحديد عناوين الذاكرة لمقاطع البرنامج.. وهم :

- Code Segment Register (CS)

ويحتوي هذا المسجل علي عنوان مقطع الكود للبرنامج في الذاكرة.

- ☒ Data Segment Register (DS)

ويحتوي هذا المسجل علي عنوان مقطع بيانات البرنامج(متغيرات ومخازن البيانات الديناميكة) في الذاكرة.

- ☒ Stack Segment Register (SS)

ويحتوي هذا المسجل علي عنوان جزء من الذاكرة يطلق عليه المكدس ..ويستخدم لتخزين البيانات كما سنري بعد قليل.

- ☒ Extra Segment Register (ES)

ويتم استخدام المسجل ES للتعامل مع مقطع إضافي ..

ويوجد مسجلين آخرين FS,GS وهما مؤشران علي بيانات إضافية خاصة بالـ Threads .. والواقع أنه لن تهمنا كثيرا مسجلات المقاطع في الكتاب الحالي.

3- مسجلات العنونة

وهي تشمل كلا من مسجلات الفهرسة والمؤشرات ومسجلات المكدس.

☒ مسجلات الفهرسة والمؤشرات Index & Pointer Registers

وهي مسجلات تستخدم في تحديد عناوين الذاكرة .. وهم :

```
esi (si)          Source index
edi (di)          Destination index
eip (ip)          Instruction pointer
```

من الممكن استخدام مسجلات المؤشرات كأغراض عامة أيضا ما عدا المسجل eip .. ولكن كل مسجل له غرض .. فمثلا المسجل esi يستخدم لتحديد المصدر لعنوان نص أو مصفوفة مثلا .. بينما المسجل edi يستخدم لتحديد الهدف لعنوان نص أو مصفوفة مثلا ..

بينما المسجل eip (أو ip في معالج16 بت) يحتوي علي عنوان التعليمة القادمة التي سيقوم المعالج بتنفيذها.

☒ مسجلات المكدس Stack Registers

المكدس أو الاستاك هو جزء في الذاكرة يتم عنونته بواسطة مسجل المؤشر esp ويستخدم للتخزين المؤقت للبيانات .. ويوجد مسجلان للمكدس فقط هما esp,ebp :

```
esp (sp)          Stack pointer
ebp (bp)          Base pointer
```

المسجل esp يحمل موضع المكدس الحالي الآن في الذاكرة .. بينما المسجل ebp يستخدم غالبا مع الدوال Functions ليكون مؤشر لمعاملات الدالة.

4- مسجل الرايات

وهو عبارة عن مسجل يحوي مجموعة من الرايات Flags .. لهذه الرايات نوعين :

- ☒ رايات حالة Status Flags
- ☒ رايات تحكم Control Flags

بالنسبة لرايات الحالة .. فهي تشير لحالة نتيجة تنفيذ تعليمة ما .. ومن أمثلة ذلك راية الصفر Zero Flag (ZF) والذي تكون قيمته بواحد أي يتم رفع الراية في حالة إذا كان النتيجة صفر لعمليات الجمع والطرح مثلا.

بينما رايات التحكم فهي تستخدم للتحكم بطريقة تنفيذ التعليمات.. ومن أمثلة ذلك راية الاتجاه Direction Flag (DF) والذي يحدد مثلا الاتجاه الذي يتم التعامل به مع البيانات في الذاكرة من البداية أم النهاية.

وسنتعرف أكثر علي الرايات في فصل التنقيح Debugging.

3 – استكمال شفرات *OPCode*

كما عرفنا من قبل أن شفرات OPCode هي تعليمات للمعالج , وهي عبارة عن نص قابل للقراءة بدلا من صف البيانات الثنائي.

وسنتناول في هذا الجزء بعض تعليمات OPCode المهمة الرئيسية ..

تعليمة Mov ☒

تستخدم هذه التعليمة لنسخ قيمة من مكان لأخر .. هذا المكان من الممكن أن يكون مسجل Register , عنوان ذاكرة Memory Location أو قيمة عددية.
صيغة تلك التعليمة كالتالي :

```
Mov Destination, Source
```

وتستطيع نسخ قيمة مسجل لأخر هكذا :

```
Mov eax,ebx
```

والتعليمة السابقة تستخدم لنسخ قيمة المسجل ebx إلي المسجل eax وحجم المسجل الهدف لابد أن يكون مناسب ليحتوي قيمة مسجل المصدر, فمثلا التعليمة التالية خطأ :

```
Mov ah,ebx
```

فالتعليمة السابقة نقوم فيها بوضع قيمة مسجل DWORD بحجم 32 بت في مسجل Byte بحجم 8 بت .. وبالتالي فالتعليمة تصبح هنا خاطئة.

ويتم تحديد عناوين الذاكرة داخل الذاكرة عن طريق الـ offset فمثلا افترض أن لدينا قيمة ولتكن 10AA20BB30CC40DD50EE فهذه القيمة ستكون مخزنة داخل الذاكرة علي هذا النحو مثلا :

Offset(h)	55	56	57	58	59	5A	5B	5C	5E	5F
Value(h)	10	AA	20	BB	30	CC	40	DD	50	EE

كل offset في الجدول السابق عبارة عن قيمة DWORD أي 32 بت.. فمثلا أول أوفست 55 عبارة عن قيمة 32 بت تمثل بالنظام السداسي عشر Hexadecimal هكذا [00000055h] وهذا الأوفست يمثل وحدة تحمل بايت وهو القيمة [10h] في الجدول السابق.. وبالتالي فهذا البايت له عنوان محدد offset,بمعني إذا قلت لك ما هو قيمة البايت ذو العنوان 5A ؟ فستقول لي البايت قيمته [CC] أي 204 عشريا.

فستقول لي البايت قيمته [CC] أي 204 عشريا.

وإذا أردنا مثلا نسخ قيمة العنوان 5A برمجيا إلي مسجل وليكن [al] ذو الحجم 8 بت أي بايت .. فسيكون الكود كالتالي :

```
mov al, Byte ptr [0000005Ah]
```

حيث قمنا في التعليمة السابقة بنسخ قيمة بحجم بايت عند عنوان الذاكرة pointer أو ptr رقم 5Ah إلي بايت المسجل [al] .. وبالتالي سيحمل المسجل [al] القيمة [CC].
وإذا أردنا نسخ قيمة 2 بايت من عند موضع الذاكرة 5A فسيكون الكود كالتالي :

```
mov ax, Word ptr [0000005Ah]
```

حيث قمنا في التعليمة السابقة بنسخ قيمة بحجم Word أي 2 بايت من عنوان الذاكرة 5A إلي المسجل ax ذو الحجم 2 بايت .. وبالتالي سيحمل المسجل [ax] القيمة [40CC] .

وإذا كنت قد لاحظت أن القيمة خزنت في المسجل ax بصورة معكوسة , وذلك لأن المسجل al السفلي Low هو أول مسجل يتم وضع البيانات به.. ثم يليه المسجل ah الأعلى High في ملء البيانات.

```
        AX
   AH        AL
```

وإذا أردنا نسخ قيمة 4 بايت من عند موضع الذاكرة 5A فسيكون الكود كالتالي :

```
mov eax, dword ptr [0000005Ah]
```

حيث قمنا في التعليمة السابقة بوضع قيمة بحجم 32 بت أي 4 بايت من عند عنوان الذاكرة 5A داخل المسجل eax ذو الحجم 4 بايت.. وبالتالي سيحمل المسجل [eax] القيمة [50DD40CC].

ومسألة تحديد الحجم المراد نسخه من موضع ذاكرة سواء byte أو Word أو DWord ليست ضرورية .. بما إنك تريد نسخ قيمة بحجم المسجل .. وبالتالي فمن الممكن أن تقول الأتي :

```
mov eax, [0000005Ah]
```

وسيقوم بنسخ قيمة بحجم المسجل eax - وهي هنا 4 بايت- من عند العنوان 5A.

الآن تطرقنا لكيفية نسخ قيمة مسجل لأخر .. وقيمة عنوان ذاكرة Memory Location لمسجل , وسنتطرق الآن لنسخ قيم عددية لمسجل , وستكون كالأتي:

```
mov eax, 1000
```

في التعليمة السابقة قمنا بنسخ القيمة 1000 إلي المسجل eax .. والفرق بينها وبين التعليمة التالية :

```
mov eax, [1000]
```

أنه في هذه التعليمة يتم استخدام الأقواس [] لإتيان القيمة عند عنوان الذاكرة 1000 .. ولكن بدون الأقواس فستصبح قيمة عددية فقط .

☒ التعليمات الحسابية ADD,SUB,MUL,DIV

وتستخدم تلك التعليمات في العمليات الحسابية الرئيسية .. وتستطيع تخمين وظيفة كل تعليمة من اسمها ..

التعليمة add تستخدم للجمع addition
التعليمة sub تستخدم للطرح subtraction
التعليمة mul تستخدم للضرب multiply
التعليمة div تستخدم للقسمة division

والصيغة العامة للتعليمة add هي :

```
add destination, source
```

والكود المنفذ يكون علي الصيغة :

```
destination = destination + source
```

فمثلا افترض أن قيمة المسجل al تساوي 3 ..وقيمة المسجل bl تساوي 5 فبعد تنفيذ تعليمة الجمع هكذا

```
add al , bl
```

فسيكون كود التنفيذ بالصيغة

```
;al = al + bl
;al =  3 + 5
;al =    8
```

والصيغ التالية لتعليمة الجمع add هي المسموح بها :

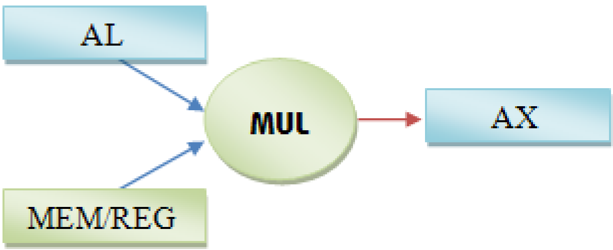
Destination	Source	Example
Register	Register	add eax , ecx
Register	Memory	add eax, dword ptr [100h] or add eax, [ecx]
Register	value	add eax, 1000
Memory	value	add dword ptr [100h], 25
Memory	Register	add dword ptr [100h], ecx

وكذلك الحال مع تعليمة الطرح sub حيث صيغة الكود لهما تشبه صيغة تعليمة الجمع add, مثلا هو موضح كالأتي :

sub destination, source   (destination = destination - source)

بينما تعليمة القسمة div وتعليمة mul مختلفين قليلا .. فيوجد تعليمات للضرب والقسمة للإشارة وبدون إشارة Signed and Unsigned .. وكذلك يتم وضع الحجم في الاعتبار ..

فعند استخدام الضرب لأرقام بحجم 8 بت .. يتم وضع الرقم الأول في المسجل AL .. والرقم الثاني يكون في مسجل أخر أو في موقع ذاكرة.. ومن ثم يكون الناتج في المسجل AX بحجم 16 بت.



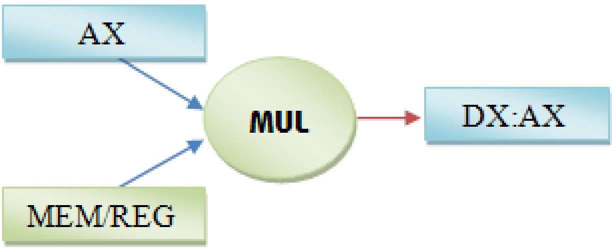
مثال :

```
x = 20
y = 30
z = x * y
z(Word) = BYTE * BYTE
```

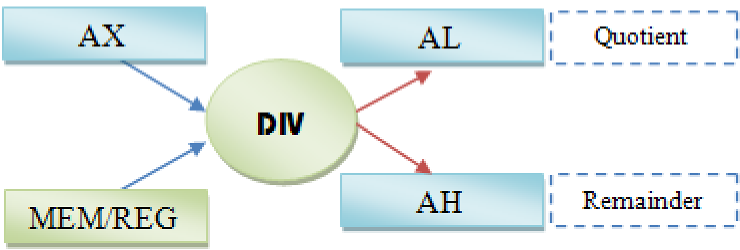
```
x db 20
y db 30
z dw ?

mov al,x
mul y
mov z, ax
```

وعند القيام بعملية ضرب بحجم 16 بت .. يتم وضع الرقم الأول في المسجل AX والرقم الثاني يكون في مسجل أخر أو موقع ذاكرة ..
ويتم تخزين القيمة الناتجة في المسجلين DX,AX بطول 32 بت.



كذلك الحال مع تعليمة القسمة DIV .. حيث يتم وضع الرقم الأول في المسجل AX .. والرقم الثاني يكون في مسجل أخر أو موقع ذاكرة .. ومن ثم تكون نتيجة القسمة في المسجل AL وباقي القسمة في المسجل AH.

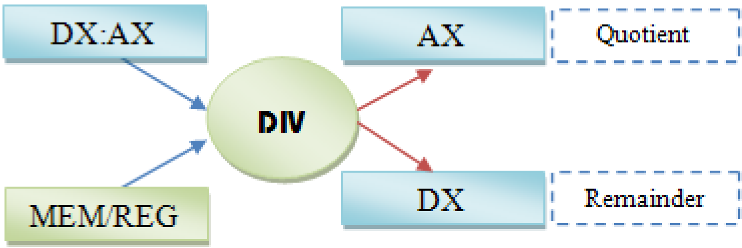


مثال :

```
x = 140
y = 25
z = x \ y
r = x mod y
```

```
x dw 140
y dw 25
z db ?
r db ?
mov ax,x
div y
mov z, al
mov r, ah
```

وعند القيام بعملية قسمة بحجم 32 بت ..



☒ تعليمتي INC,DEC

تؤدي التعليمة INC أو Increment إلي زيادة قيمة المسجل أو قيمة عند عنوان ذاكرة بمقدار الواحد.

وتؤدي التعليمة DEC أو Decrement إلي نقص قيمة المسجل أو قيمة عند عنوان ذاكرة بمقدار الواحد.

وكلا التعليمتان لهما معامل واحد فقط ..

```
;al = 25
        inc al   →   al = al + 1
```

```
;al becomes 26
        dec al  →  al = al - 1
;al becomes 25
```

inc Byte ptr [1020] → القيمة عند العنوان 1020 ستزيد واحد
dec Byte ptr [1020] → القيمة عند العنوان 1020 ستنقص واحد

☒ تعليمة Exchange

وتكتب هكذا XCHG وهي تستخدم لتبديل Swap قيمتي مسجلين أو عناوين ذاكرة ببعضهما البعض ..
مثال :

```
;al = 25
;bh = 30

xchg al,bh

;al becomes 30
;bh becomes 25
```

☒ تعليمتي POP,Push

(عودة لشرح الاستاك STACK)

عرفنا فيما سبق أن الاستاك أو المكدس عبارة عن مكان في الذاكرة يتم عنونته بمسجل مؤشر الاستاك ESP .. وأنه يستخدم لحفظ القيم المؤقتة , ويوجد تعليمتان أساسيتان لحفظ تلك القيم واسترجاعها مرة أخري , تعليمة Push وهي تدفع القيمة المراد حفظها إلي الاستاك .. وتعليمة POP وهي تسترجع القيمة من الاستاك..

وأخر قيمة يتم دفعها في الاستاك بواسطة التعليمة Push.. هي أول قيمة يتم استرجاعها من الاستاك بالتعليمة POP ..

فمثلا اعتبر الاستاك مخزن للكتب .. حيث تضع الكتب فوق بعضها في ترتيب رأسي .. من أسفل لأعلي .. فآخر كتاب تضعه في هذا المخزن .. هو أول كتاب تحصل عليه.

مثال :

```
1)  mov eax, 25
```

```
2)   mov ecx, 30
3)   push ecx
4)   push eax
5)   add eax, ecx
6)   mov edx, eax
7)   pop eax
8)   pop ebx
```

السطر الأول يتم فيه تخزين القيمة 25 في المسجل eax .

السطر الثاني يتم فيه تخزين القيمة 30 في المسجل ecx .

السطر الثالث يتم فيه دفع القيمة 30 للمسجل ecx في الاستاك.

السطر الرابع يتم فيه دفع القيمة 25 للمسجل eax في الاستاك.

السطر الخامس يتم فيه جمع قيمة المسجل ecx إلي قيمة المسجل eax وتخزين الناتج في المسجل eax .. وبالتالي فـeax سيصبح قيمته 55.

السطر السادس يتم فيه نسخ قيمة المسجل eax وهي القيمة الجديدة 55 إلي المسجل edx لتصبح قيمته 55.

السطر السابع يتم فيه استرجاع القيمة الأخيرة المخزنة في الاستاك وهي القيمة 25 ووضعها في المسجل eax مرة أخري.

السطر الثامن يتم فيه استرجاع القيمة السابقة المخزنة في الاستاك وهي القيمة 30 ووضعها في مسجل آخر وليكن ebx.

وإذا أردت معرفة ماذا يحدث تقريبا في الذاكرة عند استخدام تعليمتي Push و POP .. فإليك هذه الجداول التقريبية الموضحة لحالة الذاكرة .. فقبل استخدام تعليمة Push تكون حالة الذاكرة هكذا مثلا :

Offset(h)	55	56	57	58	59	5A	5B	5C	5D	5E
Value(h)	0	0	0	0	0	0	0	0	0	0

وإذا قمنا بتنفيذ الكود التالي .. ودفعت بقيمة 4 بايت إلي الاستاك :

```
mov eax , A1B2C3D4h
```

```
Push eax
```

فسيصبح حالة الاستاك في الذاكرة كالأتي :

Offset(h)	55	56	57	58	59	**5A**	5B	5C	5D	5E
Value(h)	0	0	0	0	0	**D4**	C3	B2	A1	0
						ESP				

وسيقف المؤشر ESP علي عنوان بداية التخزين في الاستاك وهو هنا 5A

وإذا قمنا بتنفيذ الكود التالي .. ودفعنا 4 بايت مرة أخري إلي الاستاك :

```
mov ecx , AABBCCDDh
push ecx
```

فسيتم تخزين الأربع بايتات عند العنوان [ESP – 4] أي عند [5A – 4] و وستصبح حالة الاستاك هكذا :

Offset(h)	55	**56**	57	58	59	**5A**	5B	5C	5D	5E
Value(h)	0	**DD**	CC	BB	AA	**D4**	C3	B2	A1	0
		ESP								

وعندنا نقوم بعمل استرجاع بالتعليمة POP فإننا نقوم بأخذ قيمة من الاستاك بطول 4 بايت من العنوان [56] إلي العنوان[56 + 4] ..

فمثلا إذا قمنا بتنفيذ الكود التالي :

```
POP eax
```

فسيتم استرجاع قيمة بطول 4 بايت من عند العنوان [ESP] .. أي ستعود لدينا القيمة AABBCCDDh لتخزن في المسجل eax .. وسينتقل المؤشر ESP إلي العنوان [ESP + 4] كما هو في الجدول الموضح لحالة الاستاك نتيجة لتنفيذ كود POP السابق ..

Offset(h)	55	56	57	58	59	**5A**	5B	5C	5D	5E
Value(h)	0	**DD**	CC	BB	AA	**D4**	C3	B2	A1	0
						ESP				

وعندما نستدعي تعليمة POP مرة أخري :

```
POP eax
```

فسيتم استرجاع قيمة بطول 4 بايت من عنوان مؤشر ESP وستصبح حالة الاستاك هكذا مثلا :

Offset(h)	55	56	57	58	59	5A	5B	5C	5D	5E
Value(h)	0	**DD**	CC	BB	AA	**D4**	C3	B2	A1	0
										ESP

☒ تعليمتي CALL,RET

تستخدم تعليمة Call للقفز داخل البرنامج إلي كود معين .. ومن ثم يتم الرجوع إلي السطر التالي لتعليمة القفز Call ..

بمعني أن هذه التعليمة تستخدم كما في لغات برمجة أخري كما يطلق عليه لاستعداء روتينات برمجية Subs أو Subroutines.
فمثلا يوجد لديك كود تستخدمه أكثر من مرة في البرنامج .. فالطبيعي في هذه الحالة ألا تعيد كتابة نفس الكود في كل مرة تريد تنفيذه , بل يمكنك كتابته مرة واحدة واستدعاءه كل مرة باستخدام تعليمة Call.

مثال توضيحي:

```
1)  mov eax, ecx
2)  Call FFFF
3)  mul eax,ecx
4)  ...
5)  ...
6)  call ENDME
7)  FFFF
8)  add eax, ecx
9)  div eax ,50
10) ret
11) ENDME
12) End The Program
```

المثال السابق للتوضيح فقط كيف تبدو عملية القفز والعودة .. ففي السطر الثاني يتم القفز إلي العنوان FFFF ومن ثم ينتقل التنفيذ للسطر السابع .. ومن ثم يتم عمل بعض العمليات الحسابية وعندما يصل للسطر العاشر سيجد تعليمة ret .. ومن ثم فسيتم العودة إلي السطر التالي لتعليمة القفز Call وبالتالي فسيتم العودة للسطر الثالث .. ومن ثم يتم تنفيذ بعض العمليات الحسابية .. إلي أن يصل إلي السطر السادس وبه تعليمة للقفز

Call إلي العنوان ENDME .. وبالتالي سيتم القفز إلي السطر الحادي عشر لتنفيذ الكود المطلوب ومن الممكن أن يكون كود لإنهاء البرنامج..

☒ وضع ملاحظات أو تعليقات Comments

يمكن إضافة تعليقات علي الشفرة .. تستخدم كشرح السطور .. أو لأي استخدام آخر .. بحيث لا تحسب ضمن سطور البرنامج عند ترجمته.

ولعمل ذلك نستخدم التوجيه (;).

مثال :

```
mov eax, 2
mov ecx, 4
add eax, ecx  ;now eax contains 6
```

☒ تعليمتي الإزاحة Shifting Instructions SHR,SHL

وتستخدم في إزاحة قيم ثنائية متتالية من اليمين أو اليسار .. وصيغتها العامة كالتالي :

```
SHL destination, count
SHR destination, count
```

فمثلا نفترض أن المسجل AL به القيمة العشرية 175 :

; AL = 10101111

فعند تطبيق الكود التالي :

```
SHL AL,2
```

سيتم عمل إزاحة لخانتين ليسار المسجل AL وتبديل البتات المزاحة بأصفار .. لتصبح قيمة المسجل AL تساوي التالي :

; AL = 10111100 = 188

بينما إذا قمنا باستخدام التعليمة SHR علي القيمة 175 هكذا:

```
SHR AL,2
```

فستصبح قيمة المسجل AL :

; AL = 00101011 = 43

حيث قمنا بعمل الإزاحة لخانتين جهه اليمين .

4 - القفزات الشرطية Conditional Jumps

القفزات الشرطية تمثل بعبارة IF .. ففي أثناء تنفيذ البرنامج نريد التفرع إلي تنفيذ كود معين إذا تحقق شرط ما .. ولعمل ذلك فنحن نستخدم القفزات الشرطية .. وبعض هذه القفزات موضح في الجدول التالي :

OPCode	Meaning
JA	Jump if Above
JAE	Jump if Above or Equal
JB	Jump if Below
JBE	Jump if Below or Equal
JE	Jump if Equal
JG	Jump if Greater
JGE	Jump if Greater than or Equal
JL	Jump if Less than
JLE	Jump if Less than or Equal
JMP	Jump unconditionally
JNA	Jump if Not Above
JNAE	Jump if Not Above or Equal
JNB	Jump if Not Below
JNE	Jump if Not Equal
JNG	Jump if Not Greater
JNGE	Jump if Not Greater or Equal
JNL	Jump if Not Less
JNLE	Jump if Not Less or Equal
JNZ	Jump if Not Zero
JZ	Jump if Zero

ويتم استخدام تلك الشروط مع تعليمة المقارنة cmp (compare) .. حيث يتم تمرير القيمتين المراد المقارنة بينهم لتلك الدالة ..

مثال :

```
cmp eax, ecx
cmp eax , 4
cmp 4 , eax
cmp 5 , 4
cmp eax , dword ptr [A5050505h]
```

ومن ثم يأتي الشرط بعد تعليمة cmp ليحدد العلاقة بين القيمتين هل هما متساويان أو إحداهم أكبر من الأخرى أو العكس ... الخ .

وإذا أردت معرفة ما يدور خلف هذه الشروط .. فدعنا نعود قليلا لمسجل الرايات Flags Register .. سبق أن ذكرنا أنه يحتوي علي مجموعة من الرايات Flags .. ونتيجة تنفيذ الشروط السابقة تخزن في بعض تلك الرايات .. حيث الراية تعتبر Bit .. وكل بت كما عرفنا مسبقا يستطيع حمل قيمتين أو احتمالين فقط ..إما صفر أو واحد ..

وبالتالي إذا قمنا بكتابة الكود التالي :

```
cmp eax , ecx
jz anywhere
```

فالكود السابق يقارن بين المسجل eax والمسجل ecx .. ومن ثم يتم استخدام الشرط jz أو *jump if zero* والذي يعني إذا تساوت قيمتي eax و ecx فاقفز إلي مكان ما في البرنامج ..

بمعني آخر ..التعليمة cmp تقوم بطرح قيمتي المسجل [eax - ecx] فلنفترض أن قيمة eax هي 4 وقيمة ecx كذلك 4 .. فمعني ذلك أن نتيجة التعليمة cmp ستكون صفر .. وهذه النتيجة تخزن في راية تدعي ZF أو Zero Flag .. وهذه الراية تتغير قيمتها لواحد نتيجة لتساوي القيمتين للتعليمة cmp .. وإذا لم تكن القيمتين متساويتين فستصبح الراية ZF بصفر.

ومن ثم يأتي دور الشرط jz .. وهو يتحقق من قيمة الراية ZF .. فإذا وجدها تساوي واحد .. فمعني ذلك أن الشرط متحقق وهو القفز إذا كانت نتيجة المقارنة صفر .. وبالتالي سيقفز إلي anywhere ..

والآن سنأخذ مثال عملي علي التفرع ..

```
1)   cmp eax , 0
2)   jz Sub1
3)   mov ecx , 1
4)   jmp sub2
5) Sub1:
6)   mov ecx , 2
7) Sub2:
```

السطر الأول نقوم فيه بمقارنة المسجل eax بالقيمة صفر .. فإذا كانا متساويان فستتغير قيمة الراية ZF إلي واحد ..

السطر الثاني يتم استخدام الشرط jz والذي يختبر حالة الراية ZF .. فإذا وجدها بواحد ..فسيتم القفز إلي Sub1 عند السطر الخامس وبدء تنفيذ الكود من عند السطر السادس ..وإعطاء المسجل ecx القيمة 2.. ومن ثم ينتقل التنفيذ إلي السطر السابع لتنفيذ الإجراء sub2.

وإذا وجد الراية ZF قيمتها صفر .. فسيتم تجاهل أمر القفز في السطر الثاني وسينتقل التنفيذ إلي السطر الثالث لإعطاء المسجل ecx القيمة واحد, ومن ثم سينتقل لتنفيذ تعليمة jmp الغير مشروطة في السطر الرابع .. للقفز إلي الإجراء sub2.

والمقابل للكود السابق في لغة البيسيك :

```
IF eax = 0 Then
   ecx = 2
ELSE
   ecx = 1
END IF
```

وفي لغة السي :

```
if (eax==0)
    ecx = 2;
else
    ecx = 1;
```

وجميع الشروط في الجدول السابق يمكن استخدامها بمعرفة اسمها أو من معرفة وظيفتها .. فمثلا الشرط JG يقوم بالقفز إذا كان القيمة الأول أعلي من القيمة الثانية .. وهكذا.

مميزات استخدام الأسمبلر MASM

يوفر Masm الصيغ البرمجية المميزة للغات البرمجة الأخرى .. كصيغ المقارنة Comparison والتكرار Loop .. وإجراءات للتعامل مع النصوص من بحث ومقارنة واسترجاع ودمج.

فمثلا في عمليات المقارنة يوفر لك الصيغ التالية :

```
.IF
.ELSEIF
.ELSE
.ENDIF
```

وفي عمليات التكرار يوفر لك الصيغ التالية :

```
.REPEAT, .UNTIL
.WHILE, .ENDW
```

ولاستدعاء الدوال والإجراءات يوفر الدالة Invoke والتي تسهل من كتابة تلك الاستدعاءات بصورة كبيرة .. بدلا من دفع المعاملات إلي الاستاك بتعليمة PUSH.

تعليمة IF

كما رأيت في الصفحة السابقة في كود المقارنة كنا نستخدم شروط المقارنة الموضحة في الجدول للقفز إلي مناطق في البرنامج .. وهي عملية مرهقة خاصة إذا كنت تريد كتابة شروط مركبة .. فستستخدم العديد من القفزات..

ولكن مع استخدام MASM يمكن كتابة كود المقارنة بصيغة متقدمة عبر الماكرو IF .. وإذا أردنا إعادة كتابة كود المقارنة في الصفحة السابقة باستخدام MASM فسيكون هكذا :

```
.IF eax==0
    mov ecx , 2
.ELSE
    mov ecx , 1
.ENDIF
```

وصيغة IF في الأسمبلر MASM .. قريبة الشبه بصيغة البيسيك .. مع استخدام معاملات Operators لغة السي C ..

ويمكن كتابة أكثر من شرط في صيغة واحدة هكذا :

```
.IF eax==0
    mov ecx , 1
.ELSEIF eax = 1
    mov ecx , 2
.ELSE
    mov ecx , 3
.ENDIF
```

ففي المثال السابق تم استخدام ELSEIF لعمل شرط جديد بطريقة منظمة.. وبدون القفزات المرهقة.

ويمكن كذلك استخدام الشروط المتداخلة هكذا :

```
.IF eax==1
   .IF ecx==2
       eax = 2
   .ENDIF
.ENDIF
```

ويمكن تركيب الكود السابق هكذا :

```
.IF (eax==1 && ecx==2)
       eax = 2
.ENDIF
```

وبالتالي حان الوقت لتعرف المعاملات Operators التي تستطيع استخدامها

OP	Meaning
==	is equal to
!=	is not equal to
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than to equal to
!	logical NOT
&&	logical AND
||	logical OR

تعليمة Repeat

تستخدم هذه التعليمة في تكرار مجموعة أوامر إلي أن يتحقق شرط ما .. فينتهي تنفيذ التكرار.

مثال :

```
1) mov eax , 0
2) .Repeat
3)    Inc eax
4) .Until eax == 10
```

السطر الأول يتم فيه نقل القيمة صفر إلي المسجل eax .

السطر الثاني وبه التعليمة Repeat وهي تعلن عن بدء تنفيذ تكرار.. من السطر الحالي.

السطر الثالث وبه نستخدم التعليمة inc لزيادة قيمة المسجل eax بمقدار واحد كل مره.

السطر الرابع وبه شرط الخروج من التكرار وهو أن يكون قيمة المسجل eax تساوي 10..أي ان شرط الخروج سيتحقق عندما المسجل eax يساوي 10.

ومعني ذلك أن بلوك الكود الموجود بين Repeat و Until سينفذ عشر مرات [0 To 9] .

تعليمة While

وهي تعتبر عكس تعليمة Repeat <u>في عملها</u> .. لإن التكرار مع While يستمر طالما يوجد شرط متحقق.. لكن Repeat يستمر التكرار إلي أن يتحقق الشرط.

مثال :

```
1) mov eax , 0
2) .While eax != 10
3)    Inc eax
4) .ENDW
```

السطر الأول يتم نقل القيمة صفر إلي المسجل eax.

السطر الثاني يتم استخدام تعليمة While لعمل تكرار .. وشرط استمرار التكرار هو أن تكون قيمة المسجل eax لا تساوي 10 ..

السطر الثالث نقوم بزيادة المسجل eax بمقدار الواحد.

السطر الرابع وبه تعليمة ENDW وهي تحدد نهاية جملة While أي أن الكود الموجود بين While و ENDW هو الذي سيتم تكرار تنفيذه.

وسيستمر التكرار 10 مرات [0 To 9] إلي أن تصل قيمة المسجل eax إلي 10 وسينتهي التكرار.

أي أن منطق الدورة هنا هو استمرار التكرار نتيجة تحقق شرط ما
بينما في تعليمة Repeat يستمر التكرار طالما لما يتحقق شرط ما

ويمكن استخدام Break للخروج من التكرار نتيجة لشروط داخلية داخل التكرار نفسه ..

مثال :

```
1) mov eax , 0
2) .While eax != 10
3)    Inc eax
4)      .IF eax==7
5)          .break
6)      .ENDIF
7) .ENDW
```

وعندما تصل قيمة المسجل eax إلي 7 فسيتم الخروج من التكرار .



الفصل الثالث

كتابة البرنامج الأول

1. بنية الملف المصدري Source File.
2. أساسيات برمجة Win32.
3. كتابة أول برنامج Hello World.

1- بنية الملف المصدري Source File

يتكون الملف المصدري Source File للغة الأسمبلي من أقسام Sections , هذه الأقسام هي :

- قسم الكود Code Section
- قسم البيانات Data Section
- قسم البيانات الغير مصرحة Uninitialized Data Section
- قسم الثوابت Constants Section
- قسم للمصادر Resources Section

قسم الكود Code يحتوي بالطبع علي كود البرنامج , قسم البيانات Data يحتوي علي أي بيانات نصية أو عددية تريد استخدامها في البرنامج .. ويتم تخزين هذه البيانات داخل الملف التنفيذي أي كان نوعه EXE,DLL أو غيره من ملفات الـ PE (Portable Executable), بينما قسم البيانات الغير مصرحة Uninitialized Data لا يحتوي علي بيانات أثناء التشغيل ولا تحفظ داخل الملف التنفيذي بل في الذاكرة.

قسم الثوابت Constants يحتوي علي أي بيانات نصية أو عددية ولكن للقراءة فقط ReadOnly , أي أنه لا يمكن تغيير قيمتها.

قسم المصادر Resources يحتوي علي الملفات الي نريد إدراجها للبرنامج.. ويتم إنشاء هذا القسم عن طريق برنامج لتحرير المصادر.

كيفية تعريف بعض الأقسام السابقة داخل ملف الشفرة

يتم التصريح عن الأقسام السابقة كالتالي :

```
.Data
; يتم الإعلان عن البيانات هنا

.Data?
; يتم الإعلان عن البيانات الغير المصرحة هنا

.Const
; يتم الإعلان عن الثوابت هنا
```

```
.Code
; يتم كتابة الكود هنا
```

مثال :

```
1   .data
2    D1 db 25
3    D2 db 19h
4    D3 db 11001b
5    D4 dw 10000
6    D5 dd ABCDEh
7    D6 db "A"
8    D7 db "A","B","C","D",0
9    D8 db "ABCD",0

10   .data?
11   D9 dd ?

12   .code
13   mov eax, D1
14   mov ecx, D2
15   add eax,ecx
16   mov D9, eax
```

السطر 1 تم تعريف القسم [.data] وهو يعتبر توجيه لمكان وجود مجموعة من المتغيرات [Variables] التي تحمل قيم ابتدائية مختلفة ..

فمثلا المتغير D1 يحمل بايت (db) أو (**d**ata **b**yte) قيمته 25 عشريا, المتغير D2 يحمل بايت قيمته 19h بالنظام السداسي عشر أي قيمته 25 عشريا.

المتغير D3 يحمل بايت قيمته Binary 11001 بالنظام الثنائي ..أي قيمته تساوي 25 عشريا.

المتغير D4 يحمل قيمة Word (dw) أو (**d**ata **w**ord) وهي 10000 عشريا .. أي أنه قادر علي حمل قيمة بحجم 2 بايت
بينما المتغير D5 يحمل قيمة (dd) أي (**d**ata **d**Word) أي double word اختصارا للتوجيه dWord .. وقيمته ABCDE بالنظام السداسي عشر.. ومعني ذلك أنه قادر علي حمل قيمة بحجم 4 بايت..

يمكن استخدام التوجيه db في تعريف مجموعة بايتات.. فمثلا المتغير D6 يحمل قيمة الحرف [A] في جدول الاسكي وهو 65 .. ولاحظ عندما نريد إسناد قيمة نص لمتغير فنستخدم علامتي التنصيص "" .. وهذا ما يفرق بين القيمة الرقمية والقيمة النصية ..

والمتغير D7 عبارة عن 5 بايتات .. كل بايت يحمل حرف .. بحيث يحمل المتغير D7 النص [ABCD].. وينتهي بـ[0] Null Terminating ليشير إلي نهاية النص في الذاكرة .. والصفر هنا يمثل بداية الرمز الأول في جدول الاسكي وليس القيمة صفر نفسها .. لاحظ أننا لم نضع الصفر بين علامتي تنصيص " 0", والرمز الأول في جدول الأسكي هو الـ Null .. ويجب أن تنتهي أي سلسلة نصيه بـ Null ..

ومن الممكن إسناد النص مباشرة إلي المتغير وسيتم تحديد عدد البايتات التي سيتم حجزها للنص .. مثلما فعلنا مع المتغير D8 .

وجميع القيم السابقة الموجودة في التوجيه [data.] سيتم تخزينها في الملف التنفيذي عند ترجمته وستوضع تحت قسم بنفس الاسم أيضا [data.] وسنتطرق لبنية الملف التنفيذي PE فيما بعد.

السطر 10 تم تعريف القسم [?data.] وهو يعتبر توجيه لمكان وجود مجموعة من المتغيرات [Variables] التي لا تحمل قيم ابتدائية, وسيتم وضع القيم بها من الذاكرة.

وفيه قمنا بالإعلان عن المتغير D9 بحجم dd..لاستخدامها فيما بعد في عمليات حسابية.

السطر 12 تم تعريف القسم [code.] وهو يعتبر توجيه لمكان بدء ووجود كود البرنامج..
وفيه قمنا بعمل بعض العمليات علي بعض المتغيرات التي قمنا بتعريفها مسبقا.

2- أساسيات برمجة Win32

أساس برمجة النوافذ في نظام التشغيل ويندوز يكمن في الـ API اختصارا إلي **I**nterface **P**rogramming **A**pplication .. وهو عبارة عن مجموعة إجراءات مغلفة داخل مكتبات, مثل مكتبات الـ DLLs لمكتبات User و Kernel و GDI
ويتيح لنا نظام التشغيل من أن نستدعي هذه الإجراءات الموجودة بداخل المكتبات .. حيث كل إجراء له اسم وعنوان ويتم استدعاء الإجراءات بأسمائها داخل مكتبات الـ DLLs .. وكل استدعاء لإجراء يسمي استدعاء لدالة Function.

كيفية استدعاء الدوال Functions

لكي نستخدم أي دالة API داخل مكتبة DLL .. فيجب علينا تضمين مكتبة الـ DLL أولا .. ومن ثم استدعاء الدالة المطلوبة داخل المكتبة المضمنه .

ولتضمين اي مكتبة (lib.) نستخدم التوجيه includelib **Include Lib**rary وهذا التوجيه يوجه البرنامج لتضمين مكتبة الـ DLL في عنوان معين في الذاكرة .. ليتم استخدام دوال المكتبة من قبلنا.

وفي المجمع MASM تكون الجملة كالأتي :

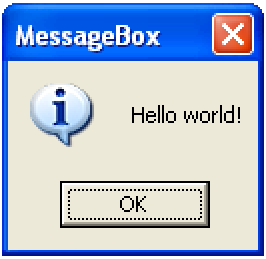
```
includelib C:\masm32\lib\user32.lib
```

علي افتراض أنك قمت بتثبيت MASM علي القرص C .. وباسم المجلد masm32 ..

بعد ذلك يجب تضمين ملف (inc.) المحتوي علي جميع تعريفات نماذج الدوال الموجودة في مكتبة الـ DLL التي قمنا بتضمينها بالتوجيه includelib

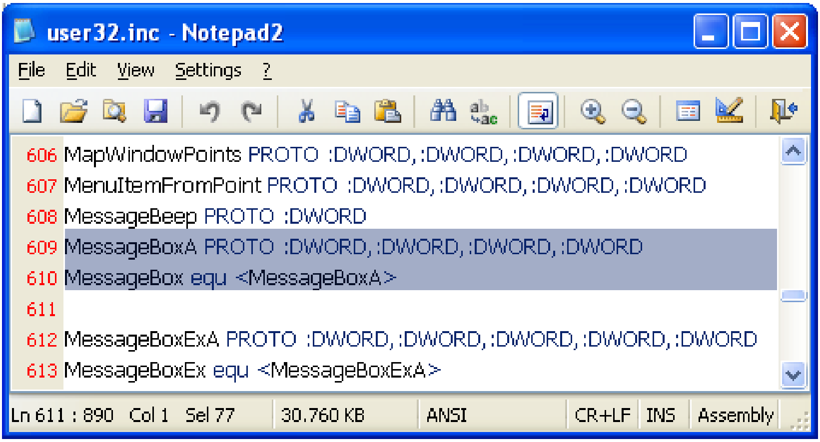
```
include C:\masm32\include\user32.inc
```

فمثلا الدالة المسئولة عن إظهار الرسائل الإرشادية في الويندوز مثل الرسالة التالية هي دالة [MessageBox] ..



وهذه الدالة [MessageBox] موجودة ضمن مكتبة user32.dll ولذلك سنقوم بتضمين هذه المكتبة بالتوجيه includelib لاستخدام هذه الدالة ..
والملف `user32.inc` به التعريف الخاص Prototype بالدالة والذي سيعتمد عليه المجمع masm .. وهو كالآتي :

```
MessageBoxA PROTO :DWORD,:DWORD,:DWORD,:DWORD
MessageBox equ <MessageBoxA>
```

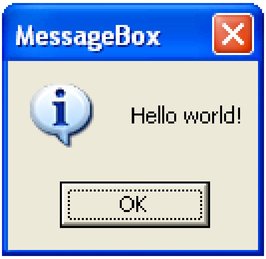


في الصورة السابقة الدالة Messagebox مساوية لتعريف الدالة MessageBoxA .. حيث تستخدم MessageBoxA مع نص **A**NSI .. حجم كل حرف 1 بايت.. بينما MessageBoxW تستخدم مع الحروف الموسعه **W**ideChar Unicode, حجم الحرف بها يكون 2 بايت .. لتمثيل جميع حروف اللغات الأخري.

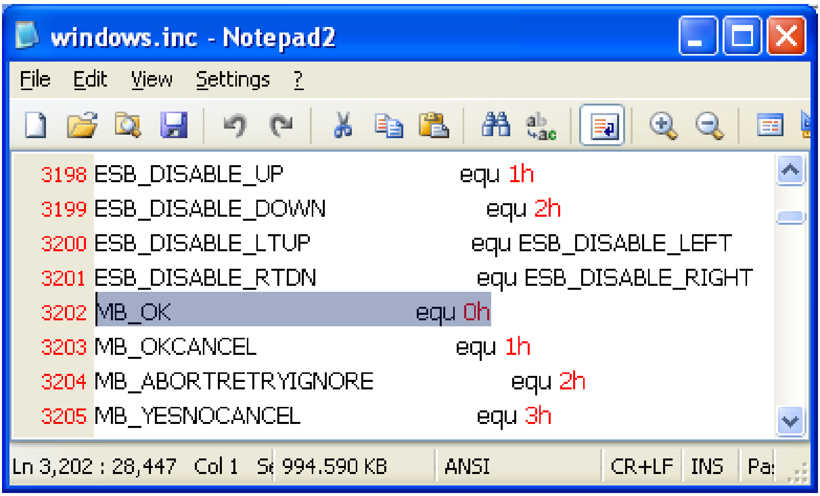
تضمين ملف الويندوز Windows.inc

الملف `windows.inc` هوه ملف خاص نقوم بتضمينه .. وهو يحتوي علي الثوابت Constants والبنيات Structures المستخدمة من قبل مكتبات النظام.

فمثلا في الصورة التالية



ستجد بها زر [OK] .. وهذا الزر له ثابت معرف داخل الملف windows.inc بالقيمة صفر تحت اسم MB_OK .. بحيث عندما نمرر هذا الثابت للدالة MessageBox فتقوم بخلق نافذة بها الزر [OK].



الصيغة العامة لكود برنامج Win32

الصيغة العامة أو الترويسة التي سنستخدمها لكتابة أي برنامج يعمل علي أنظمة Win32 .. ستكون كالتالي :

```
.386
.model flat, stdcall

option casemap:none

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc

.data

      ;Some Data Goes Here

.code

   Main:

      ;Some Code  Goes Here

   end Main
```

وفيما يلي تفسير للأسطر السابقة ..

.386

- هذا التوجيه يخبر المجمع Assembler بأن يولد كود يعمل علي معالجات 386 فأعلي.

.model flat, stdcall

- نخبر المجمع بأننا سنقوم باستخدام ذاكرة للبرنامج من النوع Flat بحجم القطاع segment يصل إلي 4 جيجا بايت .. وليس 64 كيلو بايت كما في تطبيقات 16 بت.. بمعني عندما يتم تحميل البرنامج في الذاكرة . فسيكون في قطاع واحد فقط.. وبالتأكيد سيستوعب هذا القطاع البرنامج في الذاكرة.

والتوجيه stdcall يشير إلي أن توابع الدوال المستخدمة سوف تدفع إلي الاستاك من اليمين إلي اليسار إي (أخر تابع للدالة سيدفع الأول إلي الاستاك, وأول تابع للدالة سيدفع الأخير)

ومن ثم ستقوم الدالة باستدعاء التوابع علي هذا المبدأ.

بمعني أن التوجيه stdcall يوضح كيف يتم تمرير المعاملات للإجراءات والدوال.

option casemap:none

- وهذا التوجيه يستخدم لإلغاء الفرق بين استخدام الحروف الصغيرة والكبيرة , ويفضل وضعه دائما بـــ none .

بمعني أنه مثلا عند استدعائك لـ MessageBox و messagebox فسيعتبرهما المجمع إجراءان مختلفان .. ولإلغاء هذا اللبس فنقوم بوضع الـ casemap بـ none.

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib

- كما سبق ووضحنا نقوم بتضمين المكتبتين الرئيسيتين لنظام التشغيل **ويندوز** في برنامجنا حيث سنستخدم بعض الدوال والإجراءات الموجودة بكلاهما .. وذلك سيحدث في أي برنامج نقوم بكتابته.

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc

كما سبق ووضحنا .. نقوم بتضمين ملفات التعريف للإجراءات والدوال والثوابت التي سنقوم باستخدامها تبعا للمكتبة الموجود بها هذه الإجراءات.

.data

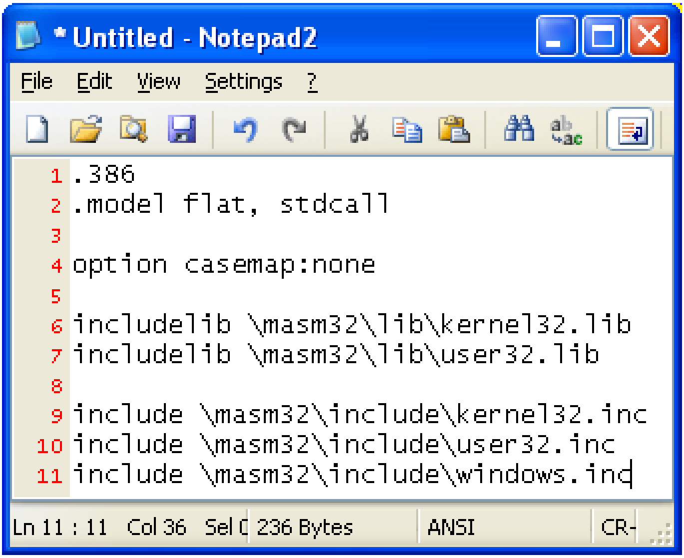
وهو توجيه لقسم البيانات والذي سنضع به أي معطيات سنستخدمها في البرنامج .. وقد سبق التطرق إليه باستفاضة.

code

وهو توجيه قسم الكود والذي سنضع به كود البرنامج.. والإجراء[:Main] يعتبر علامة علي مكان بدء الكود.. وليس شرط أن يكون بالاسمMain .. بل يمكنك أن تطلق عليه أي اسم تريده .. لكن المهم أن تكون نهاية الإجراء[:Main] بـ [end Main].

3- كتابة أول برنامج Hello World

أول خطوة تقوم بها هي إنشاء مجلد جديد علي القرصC وليكن باسم Test ثم قم بتشغيل NotePad2 أو أي محرر نصوص وضع الترويسة العامة السابقة التي سنستخدمها في أي برنامج.



ولاحظ أننا استخدمنا الـ Relative Path هكذا :

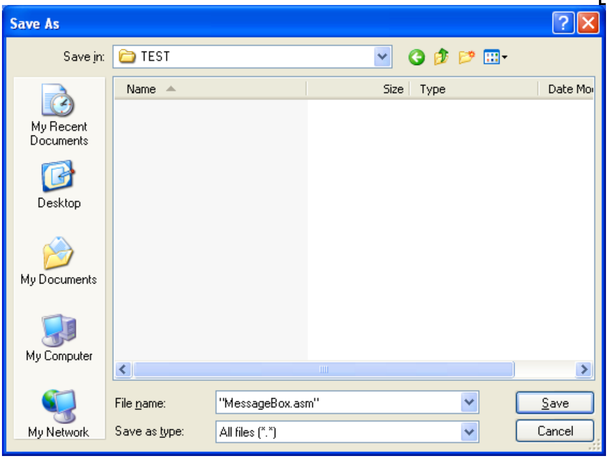
```
includelib \masm32\lib\user32.lib
```

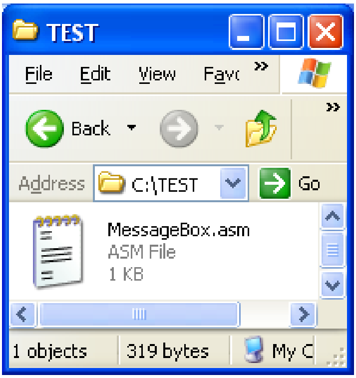
بدلا من :

```
includelib C:\masm32\lib\user32.lib
```

طالما كان كلا من المجلد Test ومجلد تثبيت MASM علي نفس القرص,وهو لدينا القرص C .

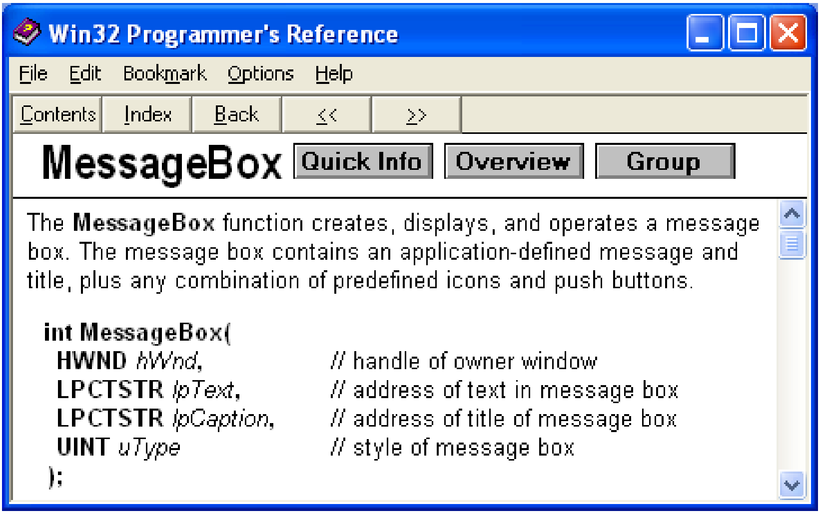
نقوم بحفظ الكود الحالي في ملف باسم "MessageBox.asm" في المجلد [C:\Test] وذلك من قائمة File واختيار Save أو Save As أو بالضغط علي [CTRL+S].





وسنقوم في هذا المثال بعرض الرسالة الشهيرة Hello World .. وسنعتمد في ذلك علي دالة MessageBox وبالتالي يلزمنا معرفة كيفية تعريف هذه الدالة وتمرير المعاملات إليها .. ولمعرفة ذلك فعليك الرجوع إلي مرجع الـ API الذي تطرقنا إليه Win32 Programmer's Reference المرفق مع هذا الكتاب تحت اسم win32.hlp

ثم اضغط علي زر Index واكتب MessageBox ليظهر لك شرح لكيفية استخدام الدالة..



The MessageBox function creates, displays, and operates a message box. The message box contains an application-defined message and title, plus any combination of predefined icons and push buttons.

```
int MessageBox(

    HWND hWnd,          // handle of owner window
    LPCTSTR lpText,     // address of text in message box
    LPCTSTR lpCaption, // address of title of message box
    UINT uType          // style of message box
   );
```

Parameters

hWnd

Identifies the owner window of the message box to be created. If this parameter is NULL, the message box has no owner window.

lpText

Points to a null-terminated string containing the message to be displayed.

lpCaption

Points to a null-terminated string used for the dialog box title. If this parameter is NULL, the default title Error is used.

uType

Specifies a set of bit flags that determine the contents and behavior of the dialog box.

حيث من الشرح السابق تستطيع أن تفهم الأتي بكل سهولة :

الدالة اسمها MessageBox وتستخدم لعرض الرسائل ولها أربع معاملات:
المعامل [hWnd] يحدد مقبض النافذة التي ستحتوي علي نافذة الرسالة.
المعامل [lpText] نضع به العنوان المحتوي علي نص الرسالة.
المعامل [lpCaption] نضع به العنوان المحتوي علي اسم نافذة الرسالة.
المعامل [uType] وهو يحدد شكل الرسالة نفسها .. حيث نعوض عنه ببعض البتات Bits تحدد خصائص النافذة .. فمثلا إذا أردت تكون الرسالة بها زر OK فنضع به الثابت MB_OK , وإذا أردت أن يكون به ok,Cancel نضع به MB_OKCANCEL .. وإذا أردت أن تحتوي نافذة الرسالة علي أيقونة التحذير فنضع به MB_ICONWARNING .. وجميع هذه الثوابت تجدها بالتفصيل ضمن مرجع الـ API

وجميع هذه الثوابت تم التصريح عنها برمجيا ضمن windows.inc والذي قمنا بتضمينه للبرنامج.

إلي الآن لا يوجد لدينا نافذة رئيسية للبرنامج وبالتالي سنعطي للمعامل hWnd القيمة صفر..

المعامل lpText سيكون مؤشر لعنوان الذاكرة للنص المراد عرضه .. كما أن المعامل lpCaption سيكون مؤشر لعنوان الذاكرة لعنوان الرسالة المراد عرضها.

سنقوم بالتصريح عن متغيرين في القسم [.data] في ملف شفرة Messagebox.asm الذي قمنا بكتابته .. كل متغير سيحمل عنوان ونص الرسالة

```
.data

MsgText db "Hello World!",0
MsgTitle db "My First MessageBox",0
```

ثم نقوم باستدعاء التعليمة invoke لدالة MessageBox في قسم الكود والتي هي من مزايا استخدامMasm32 .

```
.code

Main:

invoke MessageBox, NULL, ADDR MsgText, ADDR MsgTitle, NULL

end Main
```

حيث قمنا باستخدام التعليمه Addr للحصول علي عنوان المتغير في الذاكرة

وإذا أردت الرجوع للطريقة القديمة لاستدعاء الدوال ولم تحب استخدام دالة invoke فيجب وضع المعاملات في الاستاك .. ثم استدعاء الدالة.

```
push NULL
push offset MsgTitle
push offset MsgText
push NULL
call MessageBox
```

الآن كود البرنامج كاملا سيكون كالآتي :

```
.386
.model flat, stdcall

option casemap:none

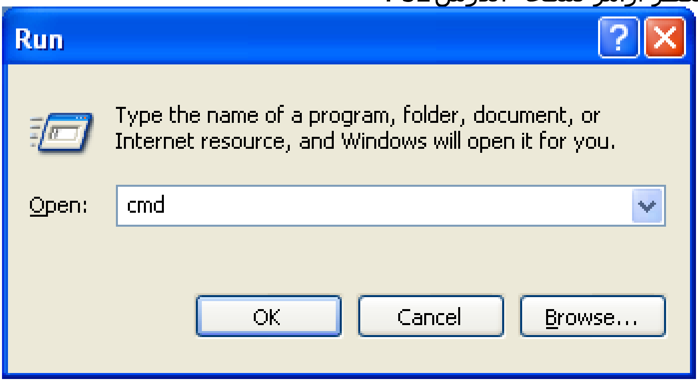
includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc

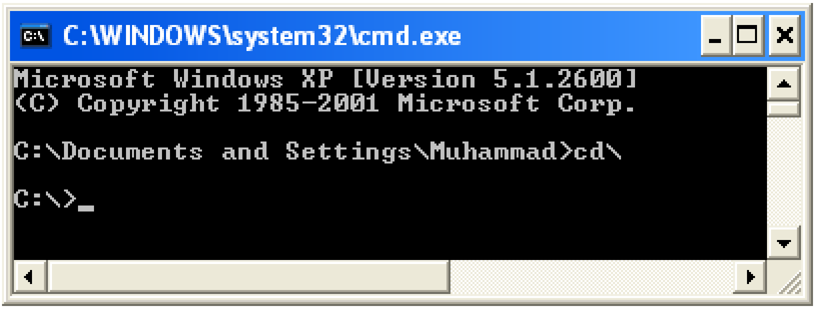
.data
      MsgText db "Hello World!",0
      MsgTitle db "My First MessageBox",0
.code
   Main:
        invoke MessageBox, NULL, ADDR MsgText, ADDR
               MsgTitle, NULL
   end Main
```

الآن سنقوم باتباع الخطوات التالية لترجمة البرنامج إلي ملف تنفيذيEXE

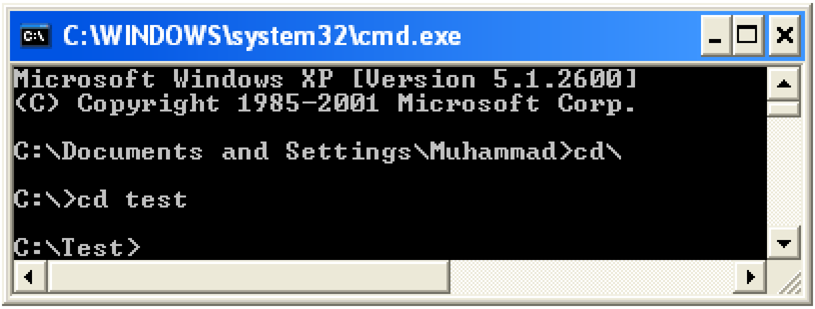
1 - من قائمة Start نقوم بتشغيل نافذة Run .. ونكتب الأمر cmd لتشغيل سطر أوامر نسخة الدوس 32 .



2 - نكتب الأمر \Cd للذهب إلي القرص سي ..



3 - نكتب الأمر Cd Test للدخول للمجلد Test



4 - نكتب الأمر ml /c /coff MessageBox.asm



حيث ml هو البرنامج الموجود في المسار تثبيت حزمة MASM32 التالي [C:\masm32\bin\ML.EXE] ..

وتوجد مرحلتين لتحويل ملف الكود إلي ملف تنفيذي ..
المرحلة الأولي وهي تجميع الكود assemble في ملف object به جميع البيانات والتعليمات في صيغة ثنائيةBinary .

المرحلة الثانية وهي الربط Link لملف الـ Object ويتم الربط بين الملف وبين عناوين المكتبات المتضمنه .. وإنشاء الملف التنفيذي تبعا لنظام التشغيل المراد العمل عليه.

في المرحلة الأولي يتم استخدام البرنامجML وهو اختصار Masm أو Macro Assembler .. وهذا البرنامج له معاملات يمكن معرفتها من خلال تمرير معامل /help للبرنامج.

C:\WINDOWS\system32\cmd.exe

```
C:\Test>ml /help
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.


        ML [ /options ] filelist [ /link linkoptions ]

/AT Enable tiny model (.COM file)         /nologo Suppress copyright message
/Bl<linker> Use alternate linker          /Sa Maximize source listing
/c Assemble without linking               /Sc Generate timings in listing
/Cp Preserve case of user identifiers     /Sf Generate first pass listing
/Cu Map all identifiers to upper case     /Sl<width> Set line width
/Cx Preserve case in publics, externs     /Sn Suppress symbol-table listing
/coff generate COFF format object file    /Sp<length> Set page length
/D<name>[=text] Define text macro         /Ss<string> Set subtitle
/EP Output preprocessed listing to stdout /St<string> Set title
/F <hex> Set stack size (bytes)           /Sx List false conditionals
/Fe<file> Name executable                 /Ta<file> Assemble non-.ASM file
/Fl[file] Generate listing                /w Same as /W0 /WX
/Fm[file] Generate map                    /WX Treat warnings as errors
/Fo<file> Name object file                /W<number> Set warning level
/FPi Generate 80x87 emulator encoding     /X Ignore INCLUDE environment path
/Fr[file] Generate limited browser info   /Zd Add line number debug info
/FR[file] Generate full browser info      /Zf Make all symbols public
/G<c|d|z> Use Pascal, C, or Stdcall calls /Zi Add symbolic debug info
/H<number> Set max external name length   /Zm Enable MASM 5.10 compatibility
/I<name> Add include path                 /Zp[n] Set structure alignment
/link <linker options and libraries>      /Zs Perform syntax check only


C:\Test>_
```

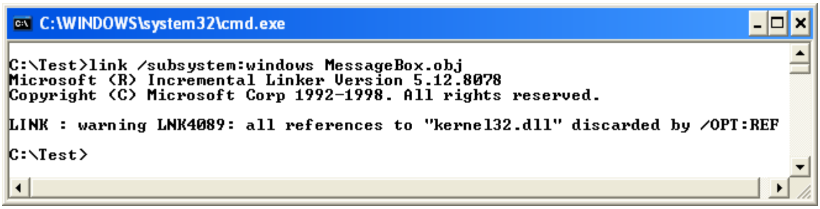
المعامل /c يؤدي إلي عدم حدوث الربطLink في المرحلة الثانية مباشرة لأننا من الممكن أن نريد استخدام خيارات أخري للربط غير الخيارات الافتراضيه.

المعامل /coff يجعل ML يولد ملف Object في صيغة COFF - Common Object File Format خاصة بالملفات التنفيذية التي تعمل تحت الويندوز.

إلي هنا انتهينا من المرحلة الأولي وخرج إلينا الملفMessageBox.obj

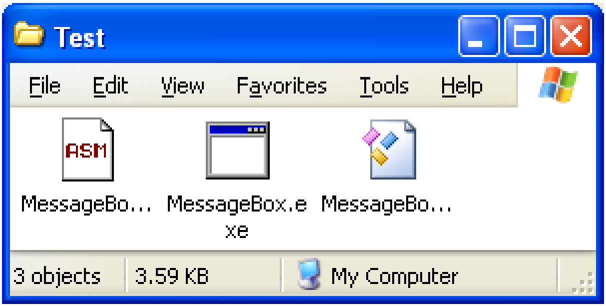
وسنقوم باستدعاء برنامج الربط Link عن طريق الأمر التالي :

link /subsystem:windows MessageBox.obj



حيث المعامل /subsystem:windows يشير للبرنامج بإنشاء ملف تنفيذي للنظام ويندوز .. وقد قمنا بتمرير ملف الـObject له..

وسيظهر لدينا الملف MessageBox.exe



وبتشغيله ستظهر رسالة Hello World التالية



وعند الضغط علي زر Ok سيعطي البرنامج خطأ Crash وهو ناتج عن عدم وضع دالة للخروج من البرنامج في نهاية التنفيذ.. ونستخدم لذلك دالة ExitProcess .. و ..ويمكنك النظر لشرح هذه الدالة في مرجع دوال الـ API .. وسيصبح الكود بعد الإضافة السابقة كالتالي :

```
.386
.model flat, stdcall

option casemap:none

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc

.data
      MsgText db "Hello World!",0
      MsgTitle db "My First MessageBox",0
.code
   Main:
        invoke MessageBox, NULL, ADDR MsgText, ADDR
               MsgTitle, NULL
        invoke ExitProcess,0
   end Main
```

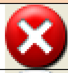
الآن ربما تريد إضافة بعد الأزرار والأيقونات للرسالة .. فيمكنك تجربة ما شئت من الجدول التالي :

Buttons Constants	
MB_OK	OK
MB_OKCANCEL	OK Cancel
MB_ABORTRETRYIGNORE	Abort Retry Ignore
MB_YESNOCANCEL	Yes No Cancel
MB_YESNO	Yes No
MB_RETRYCANCEL	Retry Cancel

يمكنك تجربة الآن تأثير أي من الثوابت السابقة علي أزرار الرسالة

```
invoke MessageBox, NULL, ADDR MsgText, ADDR
                MsgTitle, MB_YESNO
```



Icon Constants	
MB_ICONERROR MB_ICONSTOP	
MB_ICONINFORMATION	
MB_ICONQUESTION MB _ICONEXCLAMATION	
MB_ICONWARNING	

لتطبيق مثال علي الأيقونات :

```
invoke MessageBox, NULL, ADDR MsgText, ADDR
                MsgTitle, MB_ICONINFORMATION
```



مع ملاحظة أن الزر المنشط الافتراضيDefault Button دائما هو الز الأول..

من الممكن تنشيط زر أخر عن طريق الثوابت التالية :

MB_DEFBUTTON1
MB_DEFBUTTON2
MB_DEFBUTTON3
MB_DEFBUTTON4

مثلا رسالة بها زرين YesNo ونريد تنشيط زر No أولا ..

```
invoke MessageBox, NULL, ADDR MsgText, ADDR
                  MsgTitle, MB_YESNO +  MB_DEFBUTTON2
```



وبعد اختيار المستخدم للزر المطلوب تعود إحدي القيم التالية حسب الاختيار في المسجل eax :

IDOK	equ 1
IDCANCEL	equ 2
IDABORT	equ 3
IDRETRY	equ 4
IDIGNORE	equ 5
IDYES	equ 6
IDNO	equ 7
IDCLOSE	equ 8
IDHELP	equ 9



الفصل الرابع

إنشاء النافذة

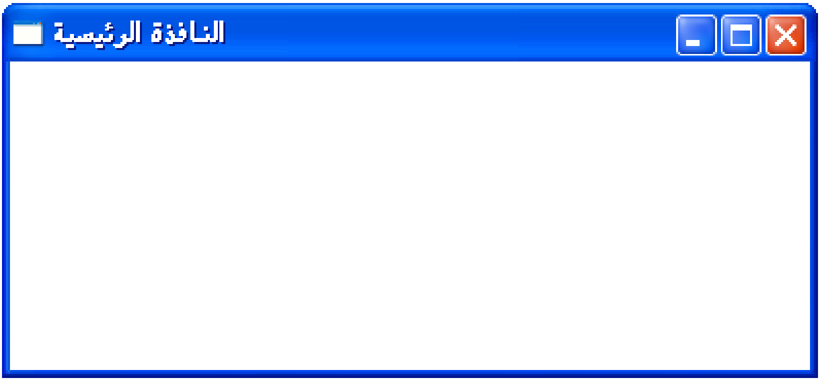
Creating the Window

1. مقدمة.
2. البنيات Structures
3. فئة النافذة Window Class.
4. تسجيل فئة النافذة Register The Window Class.
5. إنشاء النافذة Create The Window.
6. حلقة الرسائل Messages Loop.
7. إجراء النافذة Window Procedure.
8. شفرة تصميم النافذة الشاملة.

يوجد في نظام ويندوز نوعان مرئيان من البرامج .. إما تطبيقات GUI أو
تطبيقات Console .. الأولي هي المنتشرة والتي نستخدمها Graphical
User Interface .. حيث تحتوي علي واجهة مرئية للتفاعل مع المستخدم.

وكل ما تراه في نظام ويندوز عبارة عن نوافذ .. سواء واجهة النافذة الرئيسية
Parent .. ومن ثم يتم إنشاء نوافذ فرعية بتلك النافذة الرئيسية .. تشمل
جميع الأدوات مثل الزرارير وصناديق الإدخال والاختيار وهكذا ..

في الفصل الحالي سنقوم بإنشاء نافذة رئيسية مثل النافذة التالية :



ومن ثم سنقوم بمعرفة كيفية إضافة بعض الأدوات البسيطة إليها .. وحينها
يمكننا الانتقال إلي استخدام IDE لبرمجة النوافذ بعد ذلك.

2 - البنيات Structure

البنية عبارة عن مجموعه من المتغيرات .. يتم تعريف تلك المجموعه بواسطة
المعرف STRUCT هكذا :

```
Profile STRUCT
        Id              dd ?
        strName         db  20 dup (?)
        Age             db ?
Profile ENDS
```

حيث قمنا بتعريف بنية باسم Profile .. وتحتوي تلك البنية علي ثلاث متغيرات .. متغير باسم Id نوعه DWord .. ومتغير باسم Name ونوعه Byte ونقوم بحجز 20 بايت في هذا المتغير ونقوم بتهيئة الـ 20 بايت باستخدام تعليمة duplicate مع إعطائها علامة الاستفهام لكي لا يحمل المتغير قيم مسبقة .. ومتغير باسم Age ونوعه Byte.

ويتم تعريف متغيرات البنية باستخدام علامة استفهام لكي لا نعطي لها قيمة ابتدائية .

والبنية نفسها تعرف في مقدمة البرنامج قبل بداية قسم البيانات [.data] وذلك إذا كنا نريد أن تكون مجال هذه البنية عامة علي شفرة البرنامج ككل. وإلا تعرف local تحت الدوال والإجراءات.. ولا نستطيع في هذه الحالة استدعاء البنية من خارج المجال التي تم تعريفها ضمنه.

بعد ذلك يمكننا استخدام هذه البنية بعمل تهيئة لمتغير كائن وليكن باسم Student ليحمل البنية Profile هكذا :

Student Profile <>

وسيتم حجز المتغير ليشير لمتغيرات هذه البنية .. ويتم اعطاء جميع متغيرات النبية القيمة الاولية صفر.. وإذا أردنا التهيئة بقيم اولية لحظة الانشاء فيمكننا استخدام التصريح التالي:

Student Profile <1,"Amr",25>

ونستطيع الوصول في اي وقت لمتغيرات البنية وتخزين اي بيانات بها هكذا:

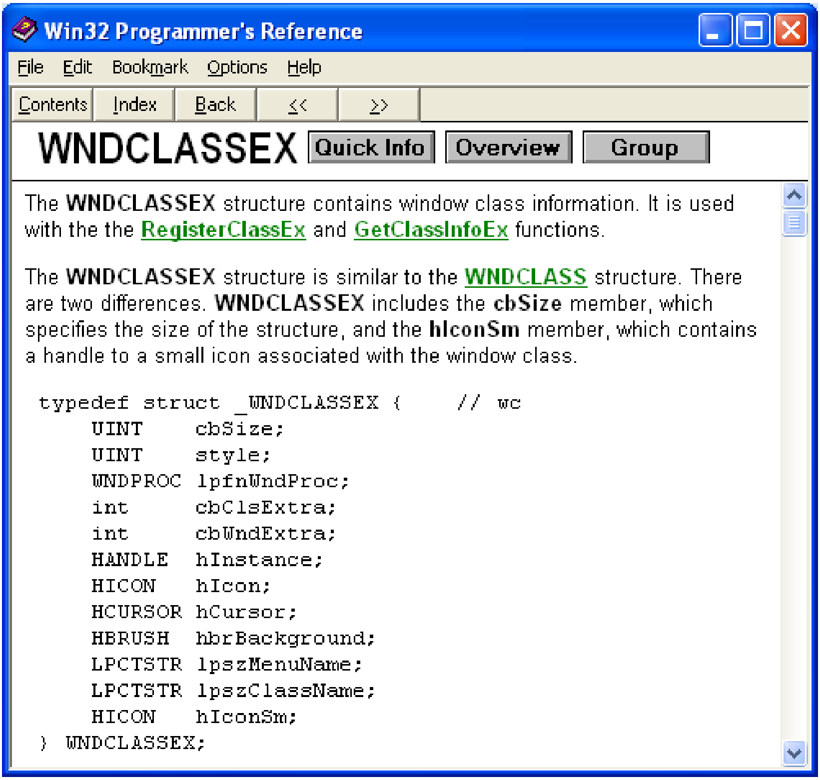
Mov Student.Id , 100
حيث قمنا باستخدام امر النسخ mov في نسخ القيمة العددية 100 إلي متغير البنية Id .

invoke lstrcpy, ADDR Student.strName, ADDR AnyName
حيث استخدمنا دالة lstrcpy في نسخ القيمة النصية عند عنوان متغير نصي وليكن باسم AnyName إلي قيمة عنوان متغير البنيةstrName.

inc Student.Id
يتم إضافة واحد إلي قيمة المتغير العدديId في بنية Student

3 - فئة النافذة Window Class

فئة النافذة WNDCLASSEX هي عبارة عن بنية تحتوي علي معلومات وخصائص النافذة المراد انشائها .. وإذا نظرت في مرجع Win32 المرفق مع الكتاب ستجد التعريف التالي لهذه البنية :



Win32 Programmer's Reference

File Edit Bookmark Options Help

Contents Index Back << >>

WNDCLASSEX Quick Info Overview Group

The **WNDCLASSEX** structure contains window class information. It is used with the the RegisterClassEx and GetClassInfoEx functions.

The **WNDCLASSEX** structure is similar to the WNDCLASS structure. There are two differences. **WNDCLASSEX** includes the **cbSize** member, which specifies the size of the structure, and the **hIconSm** member, which contains a handle to a small icon associated with the window class.

```
typedef struct _WNDCLASSEX {    // wc
    UINT    cbSize;
    UINT    style;
    WNDPROC lpfnWndProc;
    int     cbClsExtra;
    int     cbWndExtra;
    HANDLE  hInstance;
    HICON   hIcon;
    HCURSOR hCursor;
    HBRUSH  hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
    HICON   hIconSm;
} WNDCLASSEX;
```

وتشمل عناصر تلك البنية علي الأتي:

cbSize - 1

وهو معامل عددي يحمل حجم البنية WndClassEX.. ونستخدم دالة Sizeof للحصول علي حجم البنية وإعطائها لهذا المتغير.

2 - style

وهو معامل عددي يحمل قيمه تحدد سمات فئة النافذة .. مثل رسم النافذة في كل مرة يتم تغيير حجمها او تحريكها .

3 - lpfnWndProc

مؤشر إلي إجراء النافذة Window Procedure والذي يقوم بمعالجة كافة الرسائل المرسلة للنافذة .. وسنتطرق لهذا الإجراء فيما بعد.

4 - cbClsExtra

عند تعريف متغير بنية WndClassEX يقوم المترجم بتخصيص حجم ذاكرة له .. تماما كما يفعل مع جميع المتغيرات الأخري .. وإذا كنت تريد تخصيص حجم ذاكرة أكبر من الذي سيحدده المترجم فنقوم بإسناد البايتات الإضافية للمتغير cbClsExtra .. والافتراضي لهذا المتغير أن يكون بصفر.

5 - cbWndExtra

عندما يتم تشغيل البرنامج وعرضه علي الشاشة .. فهذا يعني أنه تم إنشاء نسخة Instance من البرنامج .. ويقوم نظام التشغيل بتخصيص حجم ذاكرة محدد في الرام لتقوم هذه النسخة باستخدامها .. وإذا كنت تريد تخصيص حجم ذاكره اكبر من الذي ستأخذه النسخة .. فيمكنك طلب البايتات الإضافية التي تريدها عبر هذا المتغير ليتم تخصيصها لنسخة برنامجك .. وإلا تدع نظام التشغيل يتعامل مع مسألة حجم الذاكرة لبرنامجك وتعطي المتغير القيمة صفر .

6 - hInstance

عندما تنشيء برنامج علي الويندوز فأنت تنشيء نسخة منه .. وهذه النسخة لها مقبض handle يقوم نظام التشغيل بإعطاء رقم المقبض عند إنشاء النسخة أوتوماتيكيا ..هذا الرقم يكون موحد وفريد يميز نسخة البرنامج عن البرامج الأخري .. والمعامل hInstance في فئة النافذة WNDClassEX يريد رقم المقبض لهذه النسخة .. ونستطيع الحصول علي مقبض البرنامج باستخدام دالة API تدعي GetModuleHandle.

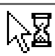
hIcon - 7

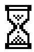
يتيح هذا المتغير استخدام ايقونه لتمثيل البرنامج .. حيث يمثل المتغير مقبض الأيقونة المراد استخدامها .. ويتيح نظام التشغيل بعض الايقونات -كما عرفنا من قبل في الفصل الثالث- والتي تأتي مع النظام والتي من الممكن استخدامها.

ID	Picture
IDI_APPLICATION	
IDI_INFORMATION	
IDI_ASTERISK	
IDI_QUESTION	
IDI_WARNING	
IDI_EXCLAMATION	
IDI_ERROR	

hCursor - 8

وهو يمثل مقبض لأيقونة تمثل مؤشر الفارة الافتراضي على نافذة البرنامج .. ويأتي مع نظام التشغيل مؤشرات يمكننا استخدامها مثل:

ID	Picture	Description
IDC_APPSTARTING		Used to show that something undetermined is going on or the application is not stable
IDC_ARROW		This standard arrow is the most commonly used cursor
IDC_CROSS		The crosshair cursor is used in various circumstances such as drawing
IDC_HELP		The combined arrow and question mark cursor is used

		when providing help on a specific item on a window object
IDC_IBEAM		The I-beam cursor is used on text-based object to show the position of the caret
IDC_SIZEALL		The four arrow cursor pointing north, south, east, and west is highly used to indicate that an object is selected or that it is ready to be moved
IDC_WAIT		The Hourglass cursor is usually used to indicate that a window or the application is not ready.

9 - hbrBackground

يحمل مقبض فرشاة تعبئة لون الخلفية .. أو احد انواع فرش النظام الافتراضية مثل COLOR_WINDOW - COLOR_BACKGROUND.

10 -lpszMenuName

إذا كنت تريد للنافذة إظهار قائمة منسدلة .. نقوم أولا بإنشاء مصدر القائمة - وسنري كيف يتم ذلك فيما بعد – ثم نقوم بتمرير مؤشر اسم هذه القائمة إلي lpszMenuName .. وإذا لم تكن تريد عرض قائمة فنقوم بتمرير القيمة Null إلي هذا المؤشر.

11 lpszClassName

مؤشر إلي نص يحدد اسم الفئة للنافذة التي تريد إنشائها .. وهذا الاسم مهم جدا تحديده .. ويمكنك كتابة اي اسم تريده ولكن اجعله مرتبط بغرض النافذة نوعا ما.

12 hIconSm

مقبض إلي الايقونة الصغير 16*16 المرتبطة بالنافذة .. والتي تظهر في شريط العنوان أو شريط المهام وغيره..

---

4 - تسجيل فئة النافذة Register The Window Class

إلي هنا سنقوم بكتابة شفرة البرنامج من بدايتها .. وسنقوم بترقيم بلوك الشفرة منعا للخلط..لأننا سنقوم بتقسيم الشفرة علي أجزاء لشرح كل جزء علي حده .. وسنقوم بوضع الشفرة في ملف وليكن باسمWindow.asm .

وأول جزء سيكون بالتأكيد التالي :

-1-

```
.386
.model flat, stdcall
option casemap:none

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc
```

سنقوم بعد ذلك بإنشاء إجراء Procedure باسم WinMain نقوم فيه بتسجيل فئة النافذة .. ومن الممكن ألا نستخدمه ولكنها تعتبر طريقة جيدة لتنظيم شفرة البرنامج .. ويلزم لإنشاء هذا الإجراء ان نقوم بتعريف صيغته العامة في بداية البرنامج هكذا :

-2-

```
WinMain PROTO STDCALL   :DWORD, :DWORD, :DWORD, :DWORD
```

```
.data?
hInstance      dd      ?

.code
Main:

    invoke  GetModuleHandle, NULL
    mov     hInstance, eax
    invoke  WinMain, hInstance, NULL, NULL, SW_SHOWNORMAL
    invoke  ExitProcess, NULL
```

حيث قمنا بانشاء Prototype للإجراء WinMain بحيث يحتوي علي 4 معاملات من نوع DWord .. مع بيان كيفية تمرير المعاملات باستخدامSTDCALL.

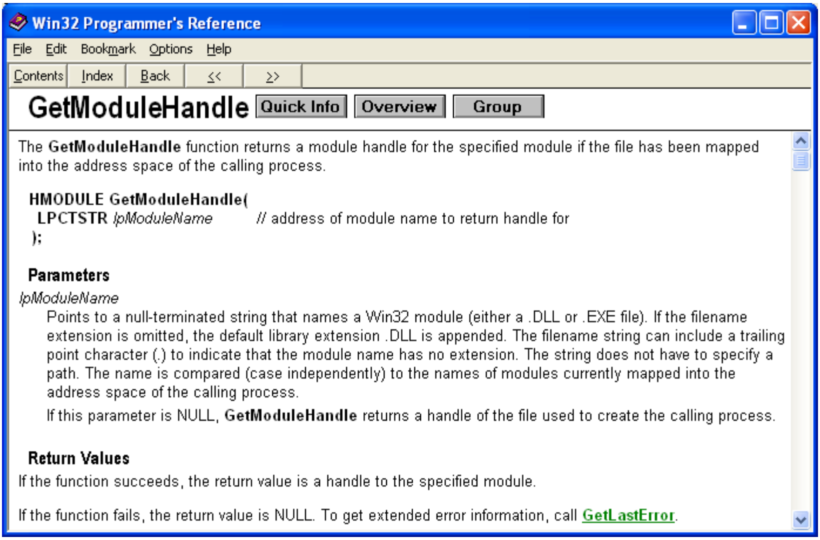
بعد ذلك قمنا بتعريف متغير نوعه dWord باسم hInstance - في قسم تصريح البيانات الغير مهيئه- ليحمل قيمة مقبض handle نسخة البرنامج والذي يعطيه النظام للبرنامج عند تشغيل النسخة.

ثم قمنا بكتابة الدالة الرئيسية Main في قسم الكود وهي اول دالة ستنفذ كما عرفنا من قبل .. ومن ثم قمنا باستدعاء دالة GetModuleHandle ومع تمرير القيمة Null لها لكي تعيد مقبض نسخة البرنامج الحالية والتي استدعت تلك الدالة.

وإذا أردنا معرفة مقبض تطبيق اخر فاننا نمرر لهذه الدالة عنوان اسم التطبيق المراد معرفة مقبضه.

وتعود الدالة بقيمة المقبض في المسجلeax ومن ثم نقوم بنسخة لدينا في المتغير hInstance لكي نستخدمه في تسجيل بنية الفئةWNDCLASSEX

ويمكنك معرفة المزيد عن دالة GetModuleHandle بالاستعانة بمرجع دوال النظام المرفق مع الكتاب ..



The **GetModuleHandle** function returns a module handle for the specified module if the file has been mapped into the address space of the calling process.

```
HMODULE GetModuleHandle(
 LPCTSTR lpModuleName       // address of module name to return handle for
);
```

Parameters

lpModuleName

Points to a null-terminated string that names a Win32 module (either a .DLL or .EXE file). If the filename extension is omitted, the default library extension .DLL is appended. The filename string can include a trailing point character (.) to indicate that the module name has no extension. The string does not have to specify a path. The name is compared (case independently) to the names of modules currently mapped into the address space of the calling process.

If this parameter is NULL, **GetModuleHandle** returns a handle of the file used to create the calling process.

Return Values

If the function succeeds, the return value is a handle to the specified module.

If the function fails, the return value is NULL. To get extended error information, call GetLastError.

ثم قمنا بعد ذلك باستدعاء الإجراء الداخلي WinMain الذي قمنا بإنشائه في البرنامج .. وقد مررنا له أربع معاملات :

المعامل الأول hInstance وهو قيمة المقبض الحالي للتطبيق ..
المعامل الثاني hPrevInstance ويستخدم في حالة إذا كنا نريد ان نعرف هل توجد نسخة اخري تعمل من البرنامج أم لا .. ولن نستخدمه حاليا .. ولذلك نمرر قيمة Null ..
المعامل الثالث lpCmdLine وهو مؤشر إلي سطر الاوامر الممرره للبرنامج عند تشغيله .. فمثلا من الممكن أن نضيف خاصية للبرنامج بأن ينفذ أوامر معينة بناء علي أمر تشغيله .. كمثال التالل

```
MyProgram.exe / h   ' show help
MyProgram.exe / u   ' Uninstall me
```

المعاملات "h/ " و "u/" تسمى معاملات اوامر ممرره للبرنامج عند تشغيله .. ومن الممكن استقبال تلك المعاملات داخل البرنامج وتنفيذ اي تعليمات تبعا للمعامل الممرر.
والمعامل الثالث هذا هو مؤشر إلي نص يحمل تلك المعاملات.. وسنري كيفية استقبال تلك المعاملات فيما بعد .. وهنا سنقوم بإعطائه قيمة Null ما دمنا لن نستخدمه .

المعامل الرابع nCmdShow وهو قيمة تشير إلي نوع حاله النافذة التي نريد إنشائها عند ظهورها علي الشاشة ومن الممكن أن ياخذ القيم التالية :

Value	Meaning
SW_HIDE	Hides the window and activates another window.
SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the system's list.
SW_RESTORE	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).
SW_SHOW	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates a window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates a window and displays it as an icon.
SW_SHOWMINNOACTIVE	Displays a window as an icon. The active window remains active.
SW_SHOWNA	Displays a window in its current state. The active window remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The active window remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

إلي الآن قمنا بالاعلان عن وجود إجراء باسم WinMain وقمنا بعمل استدعاء لهذا الإجراء .. ولكننا لما نقوم بكتابة الإجراء نفسه ولا شفرته .. والتي ستحتوي علي كودات لتسجيل فئة النافذة .. ولكتابة الإجراء فنقوم بإضافة الكود التالي إلي البلوك رقم2 .. وذلك قبل شفرة End Main.

-3-

```
WinMain proc hInst:DWORD, hPrevInst:DWORD, CmdLine:DWORD, CmdShow:DWORD

ret
WinMain endp
```

حيث قمنا بتحديد ان WinMain هو إجراء proc وقمنا بكتابة الصيغة له مع تحديد التوابع الأربع التي ستحمل القيم الممررة إليه .. حيث في الإجراءات نكتب اسم التابع ثم علامه ":" ثم نوعه .. ونفصل بين التوابع بالفاصلة المعتادة (,) ..

وفي نهاية الإجراء نكتب جملة نهاية الإجراء endp .. ونستخدم ret لإنهاء الإجراء والرجوع إلي للتعليمة التالية لتعليمة استدعاء الإجراء.

الآن يمكننا كتابة الشفرة الداخلية للإجراء WinMain والمستخدم لتسجيل فئة النافذة وبالتالي نقوم بإنشاء متغير كائن من فئة النافذة هكذا :

```
LOCAL wc:WNDCLASSEX
```

حيث قمنا بانشاء متغير فئة باسم wc من الفئة WNDCLASSEX والتعليمة Local تقوم بحجز ذاكرة للفئة wc علي الاستاك .. وستكون مدي هذه الفئة داخل الإجراء WinMain فقط .. ولا يمكن الوصول للفئة wc خارج هذا الإجراء.

إلي الآن فقط سنحتاج لمتغير نصي في قسم تصاريح البيانات Data ليحمل اسم الفئة Class name للنافذة والذي اتفقنا عليه من قبل..

وستكون شفرة البرنامج بعد الإضافات السابقة كالتالي :

-4-

```
WinMain PROTO STDCALL   :DWORD, :DWORD, :DWORD, :DWORD

.data
ClassName       db      "MyFirstClass",0

.data?
hInstance       dd      ?


.code
Main:
```

```
    invoke  GetModuleHandle, NULL
    mov     hInstance, eax
    invoke  WinMain, hInstance, NULL, NULL, SW_SHOWNORMAL
    invoke  ExitProcess, NULL

WinMain proc hInst:DWORD, hPrevInst:DWORD, CmdLine:DWORD,
             CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    mov     wc.cbSize,SIZEOF WNDCLASSEX
    mov     wc.style, CS_HREDRAW or CS_VREDRAW
    mov     wc.lpfnWndProc, OFFSET WndProc
    mov     wc.cbClsExtra,NULL
    mov     wc.cbWndExtra,NULL
    push    hInst
    pop     wc.hInstance
    mov     wc.hbrBackground,COLOR_WINDOW
    mov     wc.lpszMenuName,NULL
    mov     wc.lpszClassName,OFFSET ClassName
    invoke  LoadIcon,NULL,IDI_APPLICATION
    mov     wc.hIcon, eax
    mov     wc.hIconSm, eax
    invoke  LoadCursor,NULL,IDC_ARROW
    mov     wc.hCursor,eax
    invoke  RegisterClassEx, addr wc

    ret
 WinMain endp
end Main
```

والآن سنقوم بتحليل الأوامر الجديدة السابقة ..

mov wc.cbSize,SIZEOF WNDCLASSEX
نقوم بالحصول علي حجم البنية باستخدام المعامل SIZEOF وتمريرها للمتغير cbSize .

mov wc.style, CS_HREDRAW or CS_VREDRAW
نقوم بتمرير ثابتين لستايل النافذة .. الأول وهو CS_HREDRAW وقيمته بـ 1h ويحدد أن النافذة يعاد رسمها عند تحريكها أو تكبيرها أفقيا .. والثاني وهو

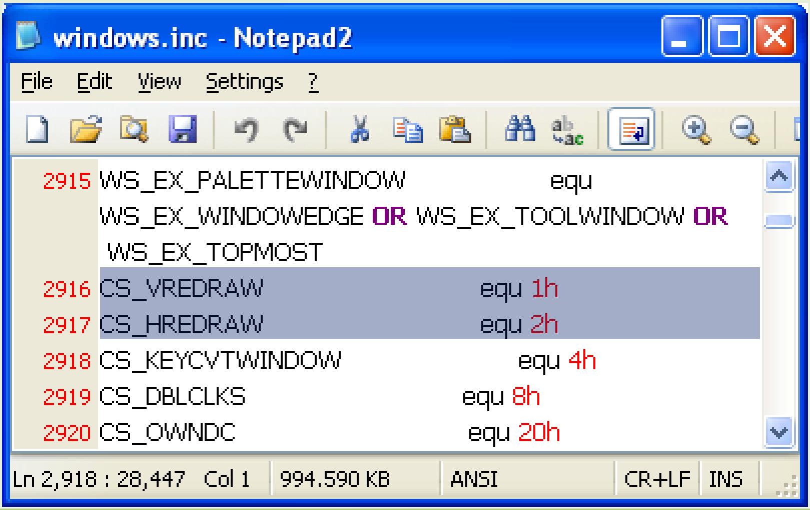
CS_VREDRAW وقيمته 2h ويحدد أن النافذة يعاد رسمها عند تحريكها أو تكبيرها رأسيا ..

وقد تم استعمال المعامل المنطقي or لضم القيمتين..

1 or 2 = 3

```
   1
Or
  10
 -----
  11
```

وجميع القيم الاخرى التي تحدد استايل النافذة مثل CS_DBLCLKS , CS_OWNDC .. فلا توجد قيمتين إذا جمعتهما يعطيان قيمة ثالثة .. وهكذا يستطيع المترجم معرفة الثوابت التي تم تمريرها ..




```
mov  wc.lpfnWndProc, OFFSET WndProc
```

نقوم بتمرير عنوان إجراء WndProc والذي تفسيره سيأتي بعد قليل.

```
mov  wc.cbClsExtra,NULL
mov  wc.cbWndExtra,NULL
```

قم بالرجوع للخلف بضع صفحات لصفحة فئة النافذة WNDCLASSEX .. وراجع وظيفة هذان المتغيران.

```
Push  hInst
pop   wc.hInstance
```

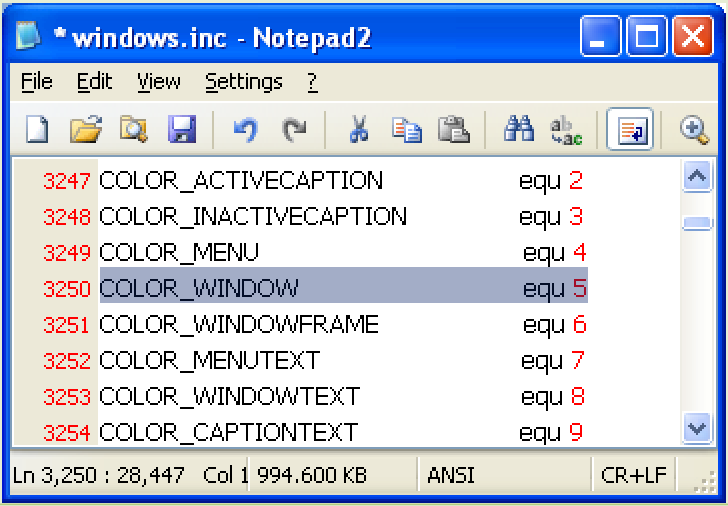
نقوم بنسخ قيمة متغير مقبض البرنامج hInst إلي معامل البنية hInstance .. ولا نستطيع أن نقول الآتي :

```
mov wc.hInstance , hInst
```

لأن تعليمة mov لا تستطيع نسخ من متغير لمتغير أخر بالذاكرة .. ويمكنك استخدام مسجل وسيط لعملية النسخ أو استخدام تعليمتى push و pop ..

```
mov   wc.hbrBackground,COLOR_WINDOW
```

نسند قيمة الثابت COLOR_WINDOW لخلفية فئة النافذة ..




```
mov  wc.lpszMenuName,NULL
```

لا توجد قائمة للنافذة حاليا فنمرر القيمة Null.

```
mov  wc.lpszClassName,OFFSET ClassName
```

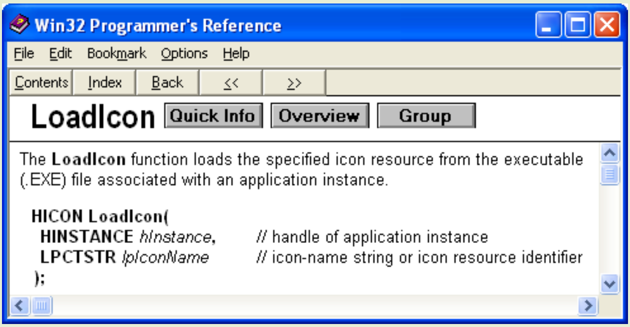
نمرر مؤشر اسم النافذة "MyFirstClass" للمؤشر lpszClassName .

```
invoke  LoadIcon,NULL,IDI_APPLICATION
```

```
mov     wc.hIcon, eax
mov     wc.hIconSm, eax
```

للحصول علي مقبض الأيقونة نقوم باستدعاء الدالة LoadIcon .. والتابع الأول لتلك الدالة وهو HINSTANCE يشير لمقبض البرنامج المراد تحميل أيقونه به .. وعند استخدام أيقونات النظام القياسيه نمرر لهذا التابع القيمة Null .. بينما التابع الثاني وهو LPCTSTR يشير لنص محتوي علي اسم الايقونه .. ويحتوي النظام علي الايقونات القياسية التالية :

IDI_APPLICATION	: ايقونه التطبيق الافتراضية
IDI_ASTERISK	: ايقونه النجمة
IDI_EXCLAMATION	: ايقونة التعجب
IDI_HAND	: ايقونة اليد
IDI_QUESTION	: ايقونة الاستفهام
IDI_WINLOGO	: ايقونه شعار النظام



وتعود الدالة بمقبض الايقونة المحمله في المسجل eax ومن ثم نمرره لمقبض الأيقونة الكبيرة hIcon والصغيرة hIconSm.

```
invoke  LoadCursor,NULL,IDC_ARROW
mov     wc.hCursor,eax
```

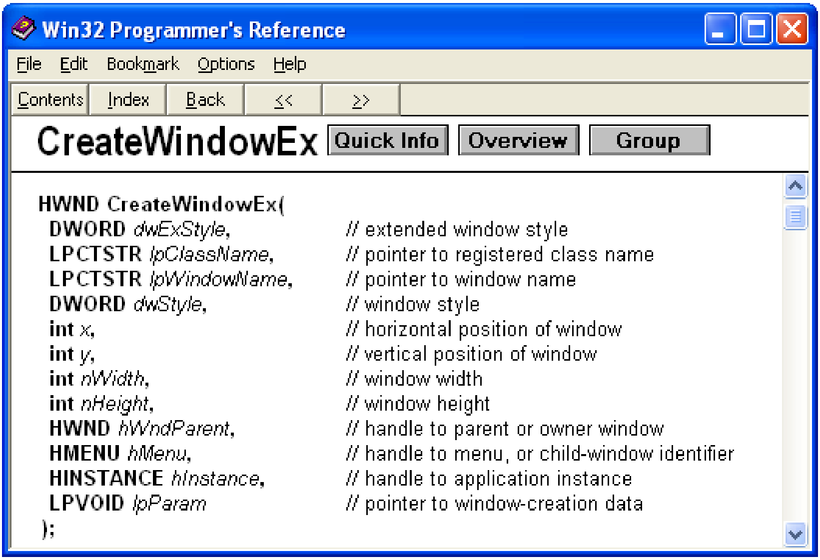
نقوم أيضا بالحصول علي مقبض مؤشر الفأرة باستدعاء الدالة LoadCursor والتي تشبه في استخدامها دالة LoadIcon.

```
invoke  RegisterClassEx, addr wc
```

اخيرا نقوم بتسجيل فئة النافذة وذلك باستدعاء الدالة RegisterClassEx وتمرير مؤشر الفئة لها.

5 - إنشاء النافذة Create The Window

بعد تسجيل فئة النافذة سنقوم باستخدام دالة تدعي CreateWindowEx لها الصيغة العامة التالية :




```
HWND CreateWindowEx(
  DWORD dwExStyle,        // extended window style
  LPCTSTR lpClassName,    // pointer to registered class name
  LPCTSTR lpWindowName,   // pointer to window name
  DWORD dwStyle,          // window style
  int x,                  // horizontal position of window
  int y,                  // vertical position of window
  int nWidth,             // window width
  int nHeight,            // window height
  HWND hWndParent,        // handle to parent or owner window
  HMENU hMenu,            // handle to menu, or child-window identifier
  HINSTANCE hInstance,    // handle to application instance
  LPVOID lpParam          // pointer to window-creation data
);
```

1 - dwExStyle

وهو متغير عددي يمثل سمات إضافية للنافذة ومن الممكن أن يحمل هذا المتغير قيم ثوابت عديدة تستطيع استكشافها بنفسك من مرجع مبرمجي Win32 الملحق مع الكتاب ..
من بعض تلك القيم قيمة تحدد اتجاه النافذة واتجاه النصوص من اليمين لليسار RTL أو من اليسار لليمين LTR .. وهي مهمه للواجهه العربية.

2 - lpClassName

مؤشر إلي اسم الفئة التي قمنا بتسجيل الفئة بها والتي في مثالنا كان الاسم "MyFirstClass" .

lpWindowName - 3

مؤشر إلي اسم النافذة الذي يظهر في عنوان النافذة Caption, وبالتالي سنحتاج إلي التصريح عن متغير ليحمل اسم عنوان النافذة .

dwStyle - 4

متغير عددي يمثل سمات إضافية للنافذة .. وستجد الثوابت التي يستطيع أن يحملها هذا المتغير في المرجع تحت دالة CreateWindow تحت قسم التعليقات Remarks.

بعض تلك الثوابت نستعرضها كالتالي :

Style	Meaning
WS_CAPTION	Creates a window that has a title bar (includes the WS_BORDER style).
WS_CHILD	Creates a child window. This style cannot be used with the WS_POPUP style.
WS_CHILDWINDOW	Same as the WS_CHILD style.
WS_DISABLED	Creates a window that is initially disabled. A disabled window cannot receive input from the user.
WS_HSCROLL	Creates a window that has a horizontal scroll bar.
WS_ICONIC	Creates a window that is initially minimized. Same as the WS_MINIMIZE style.
WS_MAXIMIZE	Creates a window that is initially maximized.
WS_MAXIMIZEBOX	Creates a window that has a Maximize button. Cannot be combined with the WS_EX_CONTEXTHELP style. The WS_SYSMENU style must also be specified.
WS_MINIMIZE	Creates a window that is initially minimized. Same as the WS_ICONIC style.
WS_MINIMIZEBOX	Creates a window that has a Minimize button. Cannot be combined with the WS_EX_CONTEXTHELP style. The WS_SYSMENU style must also be specified.

WS_OVERLAPPED	Creates an overlapped window. An overlapped window has a title bar and a border. Same as the WS_TILED style.
WS_OVERLAPPEDWINDOW	Creates an overlapped window with the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles. Same as the WS_TILEDWINDOW style.
WS_SIZEBOX	Creates a window that has a sizing border. Same as the WS_THICKFRAME style.
WS_SYSMENU	Creates a window that has a window-menu on its title bar. The WS_CAPTION style must also be specified.
WS_TILED	Creates an overlapped window. An overlapped window has a title bar and a border. Same as the WS_OVERLAPPED style.
WS_TILEDWINDOW	Creates an overlapped window with the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles. Same as the WS_OVERLAPPEDWINDOW style.
WS_VISIBLE	Creates a window that is initially visible.
WS_VSCROLL	Creates a window that has a vertical scroll bar.

x - 5

متغير عددي يحدد موضع النافذة علي المستوي الافقي بالنسبة للشاشة حيث بداية القياس من يسار الشاشة ويبدأ القياس من القيمة صفر .. وإذا قمنا بتمرير القيمة CW_USEDEFAULT فسيتم وضع النافذة أفقيا افتراضيا تبعا للنظام.

y - 6

متغير عددي يحدد موضع النافذة علي المستوي الرأسي بالنسبة للشاشة حيث بداية القياس من أعلي الشاشة ويبدأ القياس من القيمة صفر .. وإذا قمنا بتمرير القيمة CW_USEDEFAULT فسيتم وضع النافذة رأسيا افتراضيا تبعا للنظام.

7 - nWidth

متغير عددي يحدد طول عرض النافذة .. والقياس الافتراضي بوحده البيكسل.

8 - nHeight

متغير عددي يحدد طول ارتفاع النافذة .. والقياس الافتراضي بوحده البيكسل.

9 - hWndParent

متغير عددي يحمل مقبض النافذة الأب .. أو النافذة المالكه للنافذة الحالية التي نقوم بإنشائها .. ويستخدم في حالة إذا أردنا انشاء نوافذ فرعيه.

10 -hMenu

متغير عددي يحدد مقبض القائمة التي نريد استخدامها مع النافذة.. أو معرف للنوافذ التحكمات الابنه للنافذة الرئيسية.

11 -hInstance

متغير عددي يحدد مقبض البرنامج hInstance الذي حصلنا عليه بداله GetModuleHandle ليتم ربطه مع النافذة الحالية.

12 -lpParam

مؤشر يستخدم في حالة إنشاء نوافذ واجهات متعددة MDI .. Multiple Document Interface ليشير لعنوان فئة النافذة الرئيسية .. ولن نتطرق لهذا الأمر الآن ونمرر القيمة Null له .
الآن سنقوم باستدعاء دالة إنشاء النافذة CreateWindowEX داخل الشفرة وسنقوم باستدعائها بعد دالة تسجيل فئة النافذة RegisterClassEx مباشرة.

ولا ننسي أن نعرف متغير وليكن باسم AppTitle ليحمل عنوان النافذة في قسم البيانات [.data]

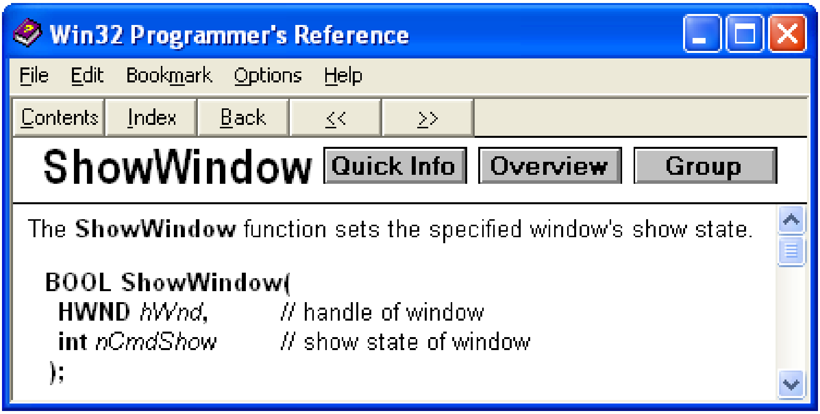
```
.data
 ClassName      db      "MyFirstClass",0
 AppTitle       db      "النافذة الرئيسية",0
```

-5-

```
INVOKE CreateWindowEx,NULL,ADDR ClassName,ADDR AppTitle , \
          WS_TILEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, \
          200,150,NULL,NULL,hInst , NULL
```

مع ملاحظة أننا يمكننا استخدام الرمز "\" إذا أردنا استكمال كود السطر علي سطر جديد..

بعد نجاح دالة CreateWindowEx ستعود بقيمة مقبض النافذة في المسجل eax هذا المقبض سنحتاج إليه لنمرره للدالة ShowWindow والتي ستقوم بإظهار النافذة علي الشاشة.



حيث **المعامل الأول** للدالة hwnd يحدد مقبض النافذة .. و**المعامل الثاني** nCmdShow يحدد نوع العملية المراد قيامها علي النافذة .. ومن الممكن أن يأخذ أي من القيم التالية :

Value	Meaning
SW_HIDE	Hides the window and activates another window.
SW_MAXIMIZE	Maximizes the specified window.
SW_MINIMIZE	Minimizes the specified window and activates the next top-level window in the Z order.
SW_RESTORE	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.
SW_SHOW	Activates the window and displays it in its current size and position.
SW_SHOWDEFAULT	Sets the show state based on the SW_ flag specified in the STARTUPINFO structure

	passed to the CreateProcess function by the program that started the application.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as a minimized window.
SW_SHOWMINNOACTIVE	Displays the window as a minimized window. The active window remains active.
SW_SHOWNA	Displays the window in its current state. The active window remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The active window remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

وسنمرر لهذا المعامل القيمة SW_SHOW لإظهار النافذة .. وسنستكمل الشفرة كما يأتي :

-6-

```
mov     hwnd,eax
invoke  ShowWindow, hwnd, SW_SHOW
invoke  UpdateWindow, hwnd
```

حيث قمنا بإتيان قيمة مقبض النافذة من المسجل eax .. وقمنا بتخزينه في متغير وليكن hwnd .. ولا تنسي الإعلان عن هذا المتغير ..
ثم قمنا بتمرير قيمة المقبض مع حالة النافذة إلي دالة ShowWindow , والنافذة الآن سيتم إظهارها علي الشاشة .. ولكن لا بأس من استدعاء دالة UpdateWindow والتي تقوم بإعادة رسم النافذة علي الشاشة .. حتي نتأكد من إظهارها .

The **UpdateWindow** function updates the client area of the specified window by sending a WM_PAINT message to the window if the window's update region is not empty. The function sends a WM_PAINT message directly to the window procedure of the specified window, bypassing the application queue. If the update region is empty, no message is sent.

```
BOOL UpdateWindow(
  HWND hWnd        // handle of window
);
```

6 - حلقة الرسائل Messages Loop

تستطيع النافذة أن تتواصل مع البرنامج ومع النوافذ الأخري عن طريق الرسائل Messages .. فكل نافذة لها حلقة غير منتهيه يتم الفحص فيها عن وجود رسائل مرسلة للنافذة .. هذه الرسائل من الممكن أن تكون رسائل لحركة الماوس علي النافذة .. ضغطات الماوس علي النافذة .. رسائل تغيير حجم وموضع النافذة رسائل تنشيط النافذة أو تصغيرها أو تكبيرها وغيرها الكثير ..

ونحن هنا نقوم في تلك الحلقة باستقبال الرسائل المرسلة إلي النافذة ومن ثم تمرير تلك الرسائل إلي إجراء النافذة Window Procedure .. وهذا الإجراء مسئول عن تحليل وفلترة تلك الرسائل ومن ثم تنفيذ الأوامر التي نريدها تبعا للرسالة المستقبلة.

شفرة جزء حلقة الرسائل سنضيفها مباشرة بعد كتلة الكود السابقة وستكون الشفرة كالتالي :

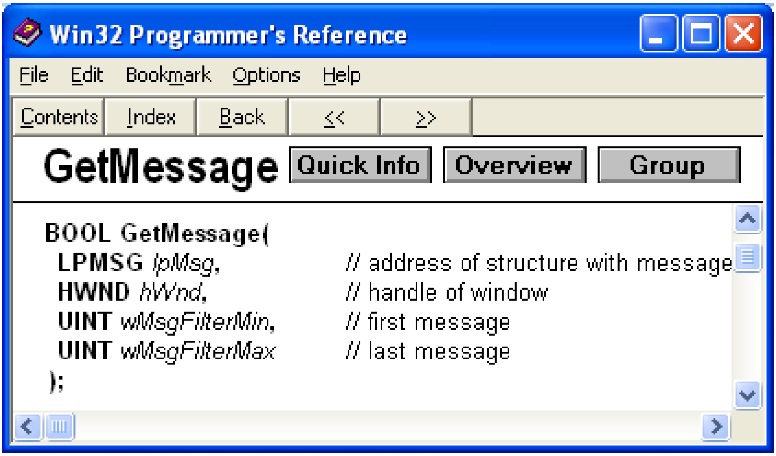
-6-

```
 .WHILE TRUE
    invoke GetMessage, ADDR msg,NULL,0,0
    .BREAK .IF (!eax)
            invoke TranslateMessage, ADDR msg
            invoke DispatchMessage, ADDR msg
.ENDW
 mov     eax,msg.wParam
```

نقوم في البداية بالإعلان عن تكرار غير منتهي بواسطة WHILE – ENDW ووضعنا شرط التقرار القيمة TRUE لكي يكون شرط التكرار دائما متحقق.

بعد ذلك نقوم باستدعاء دالة GetMessage وهي دالة استرجاع الرسالة الحاليه للنافذة ..

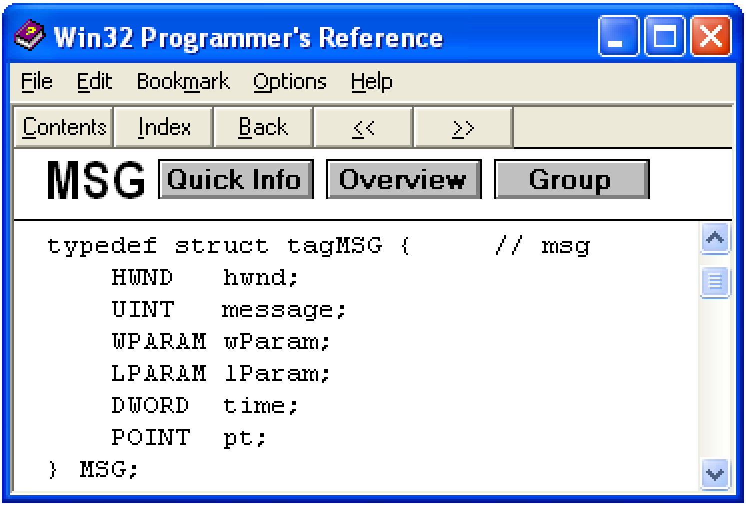
ويمكنك افتراض انه يوجد صف من الرسائل مرسلة إلي النافذة .. وكل مره يتم فيها استدعاء تلك الدالة فيتم استرجاع اول رسالة في هذا الصف .. وهكذا ...



هذه الدالة لها أربعة معاملات :

lpMSG - 1

وهو مؤشر لبنية من نوع MSG .. هذه البنية تشمل تفاصيل الرسالة. وتحتوي تلك البنية علي العناصر التالية :



- **hwnd**
  يحدد مقبض النافذة التي سيتم معالجتها

- **message**
  يحدد رقم الرسالة .. وهذا الرقم يحدد نوع الرسالة.

- **wParam**
  قيمة بحجم 32bit تحمل بيانات إضافية للرسالة علي حسب نوعها.

- **lParam**
  قيمة بحجم 32bit تحمل بيانات إضافية للرسالة علي حسب نوعها.

- **time**
  وقت إرسال الرسالة.

- **pt**
  بنية من نوع نقطه POINT تحدد x,y لموضع مؤشر الفأرة عندما تم إرسال الرسالة.

كل ما يهمنا حاليا من البنية MSG هو رقم الرسالة .. فهذا الرقم يحدد نوع الرسالة وحينها نستطيع نحن أن نعالج تلك الرسالة برمجيا.

وسنحتاج بالتأكيد للإعلان عن متغير للبنيةMSG لكي نستقبل فيه تلك الرسائل

وسنقوم بالتصريح عن هذا المتغير تحت إجراءWinMain :

```
LOCAL msg:MSG
```

حيث اسم المتغير msg ونوعه من البنية MSG .. ونقوم بتمرير للمعامل الأول في دالة GetMessage عنوان هذا المتغير[ADDR msg].

والمعامل الثاني للدالة GetMessage هو :

2 - hWnd

ويحدد هذا المعامل مقبض النافذة المراد استقبال رسائلها ..وعند تمرير القيمة Null له .. فهذا يؤدي إلي استقبال رسائل جميع النوافذ علي مسار التنفيذ الذي استدعي الدالة.. وهذه القيمة تفيد في حالة وجود أكثر من نافذة ومن ثم نسترجع جميع الرسائل ويتم معالجه تلك الرسائل كلا حسب إجراء النافذة له ..

3 - wMsgFilterMin
4 - wMsgFilterMax

يمكن تحديد المدي لقيم الرسائل المراد استرجاعها عن طريق هذين المعاملين ..

القيمة التي تعود من استدعاء دالة GetMessage تخزن في المسجل eax وإذا كانت قيمة هذا المسجل صفر .. فهذا يعني أن النافذة استقبلت رسالة الإنهاء WM_QUIT .. ويجب الخروج من التكرار وإغلاقها .. وإلا فستعود بقيمة رسالة أخري غير القيمة صفر ..

وبالتالي يمكننا ضع شرط إنهاء الحلقة التكرارية باختبار قيمة المسجل eax حيث الشرط :

```
.IF (eax)
```

يقوم بتنفيذ الأوامر في حالة إذا كان eax به قيمة غير الصفر .. حيث القيمة صفر تكون False .. وغير الصفر تكون True ..

وبالتالي عند عكس الشرط

```
.IF (!eax)
```

فسيعني تنفيذ الأوامر في حالة إذا كانeax به القيمة الصفر.

```
Not False = True
```

وسيتم الخروج من الحلقة بالتعليمة (BREAK.) وهي طريقة لكتابة الشرط علي سطر واحد .. وإلا سيكون الشرط هكذا :

```
 .IF !eax
      .BREAK
 .ENDIF
```

الآن .. إذا لم تعود دالة GetMessage بالقيمة صفر .. فهذا يعني أنه توجد رسالة للنافذة , وبالتالي نمرر هذه الرسالة .. أي نقوم بتمرير عنوان المتغير msg إلي الدالة TranslateMessage وهي دالة مسئوله عن ترجمه الرسائل المحتويه علي أرقام مفاتيح وهميه Virtual Keys إلي أحرف أسكي ASCII قابلة للقراءه .. وتستخدم من أجل الرسائل المتعلقة بضغطات المفاتيح مثل WM_KEYDOWN و WM_KEYUP وغيرهم.
وبعد أن قمنا بترجمه الرسالة .. نقوم بتمريرها إلي دالة DispatchMessage وهي الدالة المسئولة عن تمرير الرسالة إلي إجراء النافذة الذي يخص تلك الرسالة لمعالجتها ..

وإجراء النافذة Window Procedure لم نقم بكتابته بعد ..
هل تتذكر في خطوات ملء بيانات فئة النافذة .. العنصر lpfnWndProc كان يحدد عنوان إجراء النافذة ..

```
mov  wc.lpfnWndProc, OFFSET WndProc
```

ما تقوم دالة DispatchMessage بعمله .. تمرير الرسالة إلي عنوان هذا الإجراء WndProc ..

وفي هذا الإجراء سنقوم بتحليل الرسالة وتنفيذ الردود المناسبة عليها ..

اخر خطوه وهي تمرير كود الانهاء Exit Code للمسجل eax ..

```
mov  eax,msg.wParam
```

حيث توجد طرق مختلفة لإنهاء النافذة .. وكل طريقة لها كود إنهاء معين ..
لكي نعرف به الطريقة التي تم إنهاء النافذة بها .. ويخزن هذا الكود في عنصر البيانات الإضافية wParam في الرسالة.

فمثلا يمكن انهاء النافذة عن طريق زر **(X)** أو Alt+F4 أو أمر Close وفي تلك الحالة يتم تخزين القيمة صفر في wParam ..

يمكن انهاء النافذة عن طريق كود داخل برنامجنا .. وفي هذه الحالة يتم تخزين القيمة 1 في wParam ..

ويمكن الاستفادة من تلك القيمة في أمور أخري .. ولكننا لن نستفيد منها حاليا .

٧ - إجراء النافذة Window Procedure

عرفنا سابقا أن الرسائل سترسل إلي إجراء النافذة .. وإجراء النافذة سيكون كالتالي :

```
WndProc PROTO STDCALL :DWORD, :DWORD, :DWORD, :DWORD
```

الإجراء يحمل الاسم WndProc وله أربع معاملات .. أول معامل وهو hWnd وبه مقبض النافذة .. ثاني معامل وهو uMsg وبه الرسالة .. وثالث ورابع معامل هما wParam و lParam علي التوالي ..ويحملان البيانات الإضافيه للرسالة كما عرفنا سابقا ..

شفرة الإجراء كالأتي :

-7-

```
WndProc proc hWnd:DWORD, Msg:DWORD, wParam:DWORD,lParam:DWORD
   mov eax,Msg
  .IF eax==WM_CREATE
      invoke MessageBox, NULL, ADDR AppTitle, ADDR AppTitle, NULL
  .ELSEIF eax==WM_DESTROY
      invoke PostQuitMessage, NULL
  .ELSE
      invoke DefWindowProc, hWnd, Msg, wParam, lParam
  .ENDIF
   ret
 WndProc endp
end Main
```

في البداية نقوم بنسخ قيمة الرسالة للمسجل eax .. وندخله تحت مجموعه شروط .. لتحديد بعض أنواع الرسائل مثل رسالة WM_CREATE والتي تستقبلها النافذة عند لحظة إنشائها بواسطة دالة CreateWindow ..ورسالة

WM_CLOSE والتي تستقبلها النافذة عند طلب إنهائها .. ورسالة WM_DESTROY والتي تستقبلها النافذة بعد إنهائها كإشعار وبالتالي سنحتاج إلي استدعاء دالة لغلق البرنامج ..

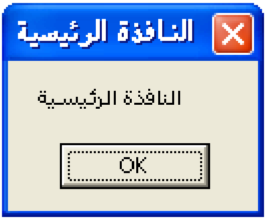
ويفضل استخدام دالة PostQuitMessage للرد علي رسالة WM_DESTROY حيث تقوم دالة PostQuitMessage بإرسال اشعار للويندوز ان مسار التنفيذ الحالي قام بإرسال طلب إنهائه .. وبالتالي عند استدعاء دالة GetMessage ستعود بالقيمة صفر وسينتهي التكرار ويعود التنفيذ للإجراء Main ويتم استدعاء ExitProcess وينتهي البرنامج.

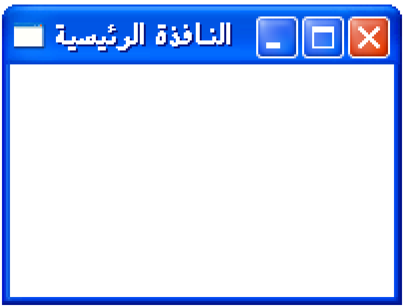
ودالة PostQuitMessage لها معامل وحيد وهو كود الإنهاء المراد من الويندوز إرساله مع رسالة WM_QUIT .. ونضع قيمته بـ NULL .

الآن نحن نقوم من خلال الإجراء WndProc باستقبال رسائل النافذة . ونعالج منها الرسائل التي نريد معالجتها .. وباقي الرسائل التي لا نريد معالجتها نمررها كما هي لإجراء النافذة الافتراضي DefWindowProc لمعالجتها.

ودالة DefWindowProc لها أربع معاملات هم نفس معاملات إجراء النافذة WndProc .. وتحرص تلك الدالة علي معالجة كافة الرسائل الممررة إليها .

إلي هنا انتهت عملية تصميم النافذة .. وبعد ترجمة الشفرة وتشغيل البرنامج ستظهر الرسالة التالية .. ثم يليها النافذة التي قمنا بتصميمها





8 - شفرة تصميم النافذة الشاملة

فيما يلي تجميع لبلوكات الشفرات السابقة الخاصة بتصميم النافذة في بلوك كود واحد .. ليسهل النظرة الشاملة للشفرة ..وستجد هذه الشفرة مرفقة مع الكتاب .

Create Simple Window

```
.386
.model flat, stdcall
option casemap:none

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc

WndProc PROTO STDCALL :DWORD, :DWORD, :DWORD, :DWORD
WinMain PROTO STDCALL   :DWORD, :DWORD, :DWORD, :DWORD

.data
AppTitle        db      "النافذة الرئيسية",0
ClassName       db      "MyFirstClass",0

.data?
hInstance       dd      ?

```

```
.code
Main:
   invoke  GetModuleHandle, NULL
   mov     hInstance, eax
   invoke  WinMain, hInstance, NULL, NULL, SW_SHOWNORMAL
   invoke  ExitProcess, NULL

 WinMain proc hInst:DWORD, hPrevInst:DWORD,
                  CmdLine:DWORD,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL hwnd:DWORD
   LOCAL msg:MSG

   mov     wc.cbSize,SIZEOF WNDCLASSEX
   mov     wc.style, CS_HREDRAW or CS_VREDRAW
   mov     wc.lpfnWndProc, OFFSET WndProc
   mov     wc.cbClsExtra,NULL
   mov     wc.cbWndExtra,NULL
   push    hInst
   pop     wc.hInstance
   mov     wc.hbrBackground,COLOR_WINDOW
   mov     wc.lpszMenuName,NULL
   mov     wc.lpszClassName,OFFSET ClassName
   invoke  LoadIcon,NULL,IDI_APPLICATION
   mov     wc.hIcon, eax
   mov     wc.hIconSm, eax
   invoke  LoadCursor,NULL,IDC_ARROW
   mov     wc.hCursor,eax
   invoke  RegisterClassEx, addr wc

  INVOKE CreateWindowEx,NULL,ADDR ClassName,ADDR AppTitle , \
           WS_TILEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, \
           200,150,NULL,NULL,hInst , NULL
```

```
mov     hwnd,eax
invoke  ShowWindow, hwnd, SW_SHOW
invoke  UpdateWindow, hwnd

.WHILE TRUE
   invoke GetMessage, ADDR msg,NULL,0,0
   .BREAK .IF (!eax)
              invoke TranslateMessage, ADDR msg
              invoke DispatchMessage, ADDR msg
.ENDW
 mov     eax,msg.wParam
 ret
WinMain endp

WndProc proc hWnd:DWORD, Msg:DWORD,
                        wParam:DWORD,lParam:DWORD
 mov eax,Msg
.IF eax==WM_CREATE
     invoke MessageBox, NULL, ADDR AppTitle, ADDR AppTitle, NULL
.ELSEIF eax==WM_DESTROY
     invoke PostQuitMessage, NULL
.ELSE
     invoke DefWindowProc, hWnd, Msg, wParam, lParam
.ENDIF
 ret
WndProc endp
end Main
```



الفصل الخامس

إنشاء نوافذ التحكم

Child Window Controls

1. مقدمة.
2. شفرة البرنامج
3. تحليل شفرة البرنامج

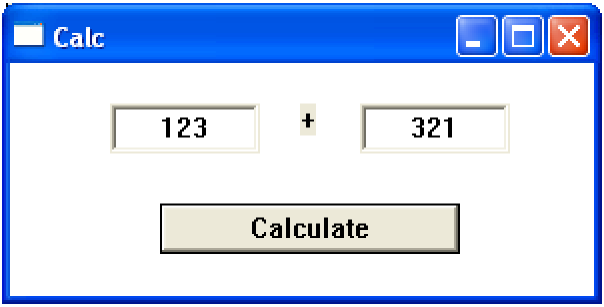
سنستعرض في هذا الفصل كيفية إنشاء نوافذ التحكم الابنه Child Controls .. والتي لا تخلو نافذة لتطبيق منها .. من أمثلة تلك النوافذ, نافذة عنصر صندوق الإدخال النصي TextBox , نافذة زر الأمر Command نوافذ أزره الاختيار مثل CheckBox و Radio وغيرها من نوافذ التحكم المختلفة.

ويوفر نظام التشغيل ويندوز فئات النوافذ السابقة .. وما علينا سوى استخدامها في تطبيقاتنا.

ونستخدم لإنشاء نوافذ التحكم الدالة CreateWindowsEx .. ولكننا لن نضطر لتسجيل فئة كل نافذة تحكم .. لإنها مسجلة من قبل النظام مسبقا.

وسنقوم بعمل برنامج يقوم بجمع رقمين يضعهما المستخدم بنفسه , وإخراج ناتج الجمع في رسالة.

نافذة البرنامج ستكون علي الشكل التالي :



حيث بها صندوقين ادخال لكلا الرقمين .. وصندوق ثابت لعلامة الزائد .. وأيضا زر جمع الرقمين والذي عند الضغط عليه ستخرج رسالة بها ناتج الجمع.

وكل نافذة تحكم قياسية لها مقبض Handle والذي نستخدمه عندما نريد التعامل مع تلك النافذة .. وأيضا لها معرف ID .. والذي يمكنا من خلال الشفرة معرفة النافذة التي نتعامل معها.

2 - شفرة البرنامج

```
.386
.model flat, stdcall
option casemap:none

includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib
includelib C:\masm32\lib\masm32.lib

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\windows.inc
include C:\masm32\include\masm32.inc

WndProc PROTO STDCALL  :DWORD, :DWORD, :DWORD, :DWORD
WinMain PROTO STDCALL :DWORD, :DWORD, :DWORD, :DWORD
CalcProc PROTO

.data
 AppTitle           db      "Calc",0
 ClassName          db      "CalcClass",0
 StaticPlusSign     db      "+",0
 EditClassName      db      "edit",0
 StaticClassName    db      "static",0
 ButtonClassName    db       "button",0
 ButtonCalcText     db       "Calculate",0

.data?
 hInstance     dd    ?
 hButtonCalc    HWND ?
 hStaticPlus    HWND ?
```

```
hEditNum1   HWND ?
hEditNum2   HWND ?
Num1Val     dd    ?
Num2Val     dd    ?
buffer      db    ?

.const
EditNum1ID   equ 1
EditNum2ID   equ 2
ButtonCalcID  equ 3

.code
Main:
   invoke  GetModuleHandle, NULL
   mov     hInstance, eax
   invoke  WinMain, hInstance, NULL, NULL, SW_SHOWNORMAL
   invoke  ExitProcess, NULL

 WinMain proc hInst:DWORD, hPrevInst:DWORD,
                    CmdLine:DWORD,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL hwnd:DWORD
   LOCAL msg:MSG

   mov     wc.cbSize,SIZEOF WNDCLASSEX
   mov     wc.style, CS_HREDRAW or CS_VREDRAW
   mov     wc.lpfnWndProc, OFFSET WndProc
   mov     wc.cbClsExtra,NULL
   mov     wc.cbWndExtra,NULL
   push    hInst
   pop     wc.hInstance
   mov     wc.hbrBackground,COLOR_WINDOW
   mov     wc.lpszMenuName,NULL
```

```
  mov     wc.lpszClassName,OFFSET ClassName
  invoke  LoadIcon,NULL,IDI_APPLICATION
  mov     wc.hIcon, eax
  mov     wc.hIconSm, eax
  invoke  LoadCursor,NULL,IDC_ARROW
  mov     wc.hCursor,eax
  invoke  RegisterClassEx, addr wc

  invoke CreateWindowEx,NULL,ADDR ClassName,\
              ADDR AppTitle , WS_TILEDWINDOW, \
              CW_USEDEFAULT, CW_USEDEFAULT, \
              300,150,NULL,NULL,hInst , NULL
 mov     hwnd,eax
 invoke  ShowWindow, hwnd, SW_SHOW
 invoke  UpdateWindow, hwnd

 .WHILE TRUE
   invoke GetMessage, ADDR msg,NULL,0,0
   .BREAK .IF (!eax)
             invoke TranslateMessage, ADDR msg
             invoke DispatchMessage, ADDR msg
 .ENDW
  mov     eax,msg.wParam
  ret
 WinMain endp

 WndProc proc hWnd:DWORD, Msg:DWORD,
                      wParam:DWORD,lParam:DWORD
  mov eax,Msg
 .IF eax==WM_CREATE
     invoke CreateWindowEx,WS_EX_CLIENTEDGE,
                  ADDR EditClassName,NULL,WS_CHILD \
                   or WS_VISIBLE or WS_BORDER or \
```

```
                ES_CENTER or ES_AUTOHSCROLL, \
                50,20,75,25, hWnd,EditNum1ID,\
                hInstance,NULL
    mov hEditNum1,eax

    invoke CreateWindowEx,NULL, ADDR StaticClassName,\
               NULL,WS_CHILD or WS_VISIBLE or \
               ES_CENTER or SS_SIMPLE , \
               145,20,10,25,hWnd,NULL, \
               hInstance,NULL
    mov hStaticPlus,eax
    invoke SetWindowText,hStaticPlus,ADDR StaticPlusSign

    invoke CreateWindowEx,WS_EX_CLIENTEDGE, \
           ADDR EditClassName, NULL,WS_CHILD \
           or WS_VISIBLE or WS_BORDER or \
           ES_CENTER or ES_AUTOHSCROLL,\
           175,20,75,25,hWnd,EditNum2ID,
           hInstance,NULL
    mov hEditNum2,eax

    invoke CreateWindowEx,NULL, ADDR ButtonClassName,\
           ADDR ButtonCalcText,WS_CHILD or WS_VISIBLE \
            or BS_DEFPUSHBUTTON,75,70,150,25,\
            hWnd,ButtonCalcID,hInstance,NULL
    mov hButtonCalc,eax

.ELSEIF eax==WM_DESTROY
     invoke PostQuitMessage, NULL
.ELSEIF Msg ==  WM_COMMAND
     mov eax,wParam
       .IF lParam != 0
          .IF ax==ButtonCalcID
```

```
                    shr eax,16
                    .IF ax==BN_CLICKED
                        invoke CalcProc
                    .ENDIF
                .ENDIF
            .ENDIF
.ELSE
    invoke DefWindowProc, hWnd, Msg, wParam, lParam
.ENDIF
 ret
WndProc endp

CalcProc proc
        invoke GetWindowText,hEditNum1,ADDR buffer,5
        invoke atol,ADDR buffer
        mov Num1Val,eax
        invoke GetWindowText,hEditNum2,ADDR buffer,5
        invoke atol,ADDR buffer
        mov Num2Val,eax
        mov eax,[Num1Val]
        add eax,[Num2Val]
        invoke dwtoa ,eax,ADDR buffer
        invoke MessageBox,NULL,ADDR buffer,ADDR AppTitle,\
                                    MB_OK
        ret
CalcProc endp

end Main
```

2 - تحليل شفرة البرنامج

في البداية قمنا بإضافة مكتبة masm32 وهي تحتوي علي دوال للتحويل النصي والعددي Numeric,Text.

السطر **16** : قمنا بالتصريح عن إجراء باسم CalcProc وفيه سنقوم بحساب عملية الجمع.

السطر **21** : وفيه حجزنا متغير لحرف الجمع الذي سيظهر علي نافذة البرنامج بين صندوقي الإدخال.

السطر **22** : وفيه حجزنا متغير لاسم فئة صندوق الإدخال واسم الفئة هو Edit .. وبالتالي عند بناء نافذة من فئة Edit فهذا يعني اننا نبني نافذة صندوق إدخال نصي.

السطر **23** : قمنا بحجز متغير لاسم فئة عنوان التعريف label .. واسم الفئة هو static .. وهو عبارة عن نافذة تحتوي علي نص ولكنها ثابتة .. أي لا يسمح بإدخال نص إليها يدويا.. وتفيد غالبا في حالة الشرح وبيان فائده ووظيفة نوافذ التحكم الأخري لمستخدم البرنامج .. وقد تم استخدامها هنا لعرض حرف الجمع [+] .. وبالتالي نعرف المستخدم ان العملية هي عملية جمع.

السطر **24** : وفيه نحجز متغير لاسم فئة زر الأمر button.. وسنستخدم تلك الفئة بالطبع في إنشاء الزرCalculate.

السطر **25** : وفيه نحجز متغير به الاسمCalculate الذي سيظهر علي الزر.

السطر **29** : نقوم بحجز متغير عددي غير مهيّأ ليحمل مقبض زر الأمر Calculate.

السطر **30** : نحجز متغير عددي لحمل مقبض نافذة حرف الجمع.

السطر **31** : نحجز متغير لحمل مقبض نافذة صندوق إدخال الرقم الأول.

السطر **32** : نحجز متغير لحمل مقبض نافذة صندوق إدخال الرقم الثاني.

السطر **33** : نحجز متغير لحمل قيمة صندوق إدخال الرقم الأول.

السطر **34** : نحجز متغير لحمل قيمة صندوق إدخال الرقم الثاني.

السطر **35** : نقوم بحجز متغير يستخدم كمخزن نصي مؤقت لتخزين قيمة كلا من صندوقي النص .. قبل تحويل تلك القيم النصية إلي قيم عددية ومن ثم تخزينها في Num1Val و Num2Val.

السطور **37** - **40** : نضع توجيه للقيم الثابتة في البرنامج .. ويستخدم التوجيه const لهذا الغرض.. والقيم الثابتة التي سنستخدمها هي قيم المعرفات IDs لكلا من نوافذ التحكم الموجودة لدينا.
حيث قمنا بالإعلان عن قيمة معرفة نافذة صندوق إدخال النص الأول EditNum1ID وحجزنا له الثابت 1 .. كذلك حجزنا لمعرف صندوق النص الثاني القيمة 2 .. ولمعرف نافذة زر الأمر Calculate القيمة 3.

السطر **93** : عند استقبال رسالة WM_Create فهذا يعني أن النافذة تم إنشائها .. ومن ثم نبدأ بإنشاء نوافذ التحكم الابنه عليها.

السطر **94** : نقوم بإنشاء نافذة صندوق الإدخال النصي الأول .. لاحظ الفئة المنشئة منها وهي edit والممرره قيمتها في المعامل الثاني .. وكذلك تمرير معرف النافذة EditNum1ID .. مع ضبط خصائص الموضع والحجم .

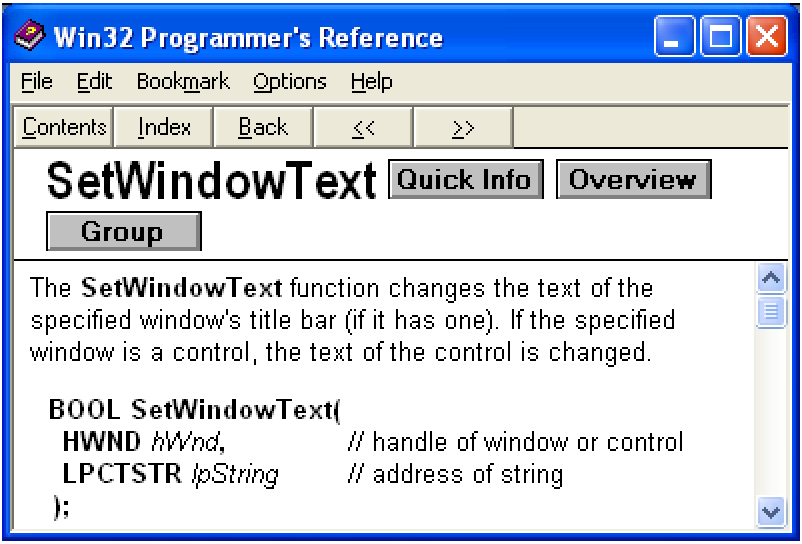
السطر **100**: بعد إنشاء نافذة تحكم صندوق الإدخال .. يتم وضع قيمة مقبض هذه النافذة في المسجل eax وبالتالي نحتاج لأن نخزنها لدينا في المتغير المناسب لذلك hEditNum1.

السطر **102**: نقوم بإنشاء نافذة علامة الجمع [+] ولا نحتاج لمعرف ID لتلك النافذة لاننا لن ندخلها في استخدامات في البرنامج بعد ذلك.

السطر **107**: نقوم بتخزين مقبض النافذة Static لدينا في المتغير المناسب .. تمهيدا لوضع علامة الجمع بتلك النافذة.

السطر **108**: نقوم باستدعاء دالة SetWindowText .. والتي تستخدم لتغيير اسم النافذة ..

وللدالة معاملان ..الأول وهو مقبض النافذة المراد تغيير اسمها .. والثاني وهو مؤشر للنص المراد تغيير اسم النافذة به.



السطر **110** : نقوم بإنشاء نافذة صندوق الإدخال الثاني.

السطر **116** : نخزن مقبض تلك النافذة لدينا في hEditNum2.

السطر **118** : نقوم بإنشاء نافذة زر الأمر من الفئة button.

السطر **122** : نخزن مقبض نافذة زر الأمر لدينا في hButtonCalc.

السطر **126** : عند استقبال الرسالة WM_Command .. فهذا يعني أن هناك إشعار وصل إلي النافذة الرئيسية لدينا .. وبالتالي سنحتاج إلي تحديد النافذة المرسل إليها عن طريق رقم المعرف الخاص بها .. ومعرفة نوع الإشعار.

```
WM_INITDIALOG                    equ 110h
WM_COMMAND                       equ 111h
WM_SYSCOMMAND                    equ 112h
WM_TIMER                         equ 113h
```

وفي حالة إذا كانت الرسالة تخص نافذة تحكم .. فالنظام يرسل إشعار WM_Command مع بيانات إضافية مثل معرف نافذة التحكم في المعامل wParam وكذلك مقبض نافذة التحكم في lParam.

وبالتالي يجب أن نتأكد أن الاشعار يخص نافذة تحكم أولا وليس من قائمة Menu في البرنامج أو من جراء استخدام Accelerator .. وذلك بمقارنه مقبض النافذة للقيمة صفر .. لإن أوامر القوائم المنسدلة ترسل إشعار WM_Command هي الأخر مع قيم صفرية لكلا المعاملين ..

وكذلك استخدام ضربات المفاتيح السريعه Accelerator مثل ALT+F4 , F1 , PRINT SCREEN,ALT+TAB تجعل النظام يرسل إشعار WM_Command إلي النافذة .. مع وضع القيمة واحد للمعامل wParam وصفر للمعامل lParam

فإذا كان lParam لا يساوي الصفر فهذا يعني أن الرسالة تخص نافذة تحكم موجودة لدينا.

ولا يفضل استخدام wParam في هذه المقارنة لأنه من الممكن أن يكون لديك معرف لنافذة تحكم بالقيمة صفر مثل حالة معرف صندوق الإدخال الأول لدينا.. وكذلك من الممكن أن يتم استخدام Accelerator .. وتحسبها علي أنها رسالة لنافذة تحكم.

المهم الآن أنه لدينا معلومتين في المعامل wParam .. هذا المعامل بطول 32 بت .. ومقسم إلي High Word و Low Word.

كلا منهم بطول 16 بت .. ويحمل معلومة مختلفة .. حيث ال Low Word والممثل بأول 16 بت .. يخزن به معرف نافذة التحكم ID الذي قمنا بوضعه عند إنشاء النافذة.

بينما الـ High Word والممثل بأخر 16 بت .. يخزن به كود الإشعار الذي تم إرسال WM_Command علي أساسه.

الآن بعد أن فهمنا الجزء السابق يمكننا استعراض سطور الشفرة مفصلة.

السطر **127** : نقوم بنقل قيمة المعامل wParam إلي المسجل eax.

السطر **128** : نقوم فيه بمقارنة المعامل lParam بالقيمة صفر , للتأكد من أن الاشعار يخص نافذة تحكم.

السطر **129** : نقوم فيه بمقارنة أول 16 بت Low Word بقيمة معرف زر Calc

وبالتالي اذا تحقق الشرط فهذا يعني أن النافذة المقصود بها الإشعار هي نافذة تحكم زر الأمر Calc.

السطر **130** : يتبقي لدينا معرفة نوع الإشعار .. وبالتالي نريد معرفة القيمة الموجودة في الـ 16 بت الأخيرة High Word .. وبالتالي سيلزم عمل إزاحة بمسافة 16 بت جهه اليمين عن طريق التعليمةSHR.

السطر **131** : ثم نعيد مقارنة قيمة المسجل بكود اشعار الحدث الذي نتوقعه وننتظره .. وهو حدث الضغط علي الزر button Click لنبدأ بحساب ناتج عملية الجمع .

```
BS_RIGHTBUTTON          equ BS_LEFTTEXT
BN_CLICKED              equ 0
BN_PAINT                equ 1
```

السطر **132** : نقوم باستدعاء إجراء CalcProc .. وكان من الممكن وضع كود إجراء CalcProc في نفس الموضع .. ولكن تنظيم البرنامج وتقسيمه إلي إجراءات فرعية لجميع أجزاء البرنامج يسهل عمليه قراءة الكود وفهمه وتصحيحه من الأخطاء.

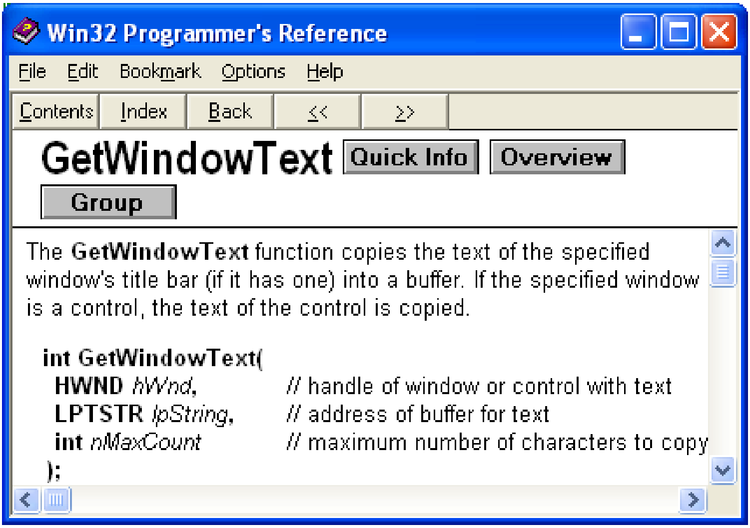
السطر **142** : بداية الإجراءCalcProc.

السطر **143** : نستخدم دالة GetWindowText لاسترجاع النص من نافذة صندوق الإدخال الأول.

حيث يوجد للدالة 3 معاملات ..

الأول وهو مقبض النافذة المراد أخذ النص منها.
الثاني وهو مؤشر للمتغير الذي ستضع به الدالة النص المعاد.
الثالث وهو طول النص المراد نسخه.

وقمنا بتمرير مقبض نافذة الإدخال الأولي hEditNum1 .. ومن ثم عنوان المتغير المخزن buffer .. ومن ثم بحجم طول بيانات يساوي 5 بايتات والبايت الأخير لغلق النص بالقيمةNull.



السطر **144** : نستدعي دالة atol والتي تقوم بتحويل من آسكي ascii الي قيمة long integer بحجم 32 بت.. فقط نقوم بتمرير عنوان النص لتلك الدالة وستعود بالقيمة العددية لهذا النص في المسجلeax.

السطر **145**: نقوم بنسخ القيمة المعادة إلي المتغيرNum1Val.

السطر **146** : نقوم بإتيان النص الموجود في نافذة الإدخال الثانية.

السطر **147** : نقوم بتحويل النص إلي قيمة عددية.

السطر **148** : نخزن القيمة العددية في المتغيرNum2Val

السطر **149** : نقوم بنسخ قيمة المتغيرNum1Val إلي المسجل eax.

السطر **150** : نقوم بإضافة قيمة المتغيرNum2Val إلي المسجل eax.

السطر **151**: نستدعي الدالة dwtoa والتي تقوم بالعكس .. اي تحول القيمة العددية dWord بطول 32 بت إلي قيمة نصية آسكي.

حيث تستقبل في المعامل الاول .. القيمة المراد تحويلها .. وفي المعامل الثاني عنوان المتغير الذي سيحمل النص المعاد من تحويل القيمة.

السطر **152** : نقوم به بعرض الرسالة بناتج عملية الجمع .

السطر **154** : نعود به إلي إجراء النافذةWndProc مرة أخري.

والآن عند تشغيل البرنامج وتجربته فستخرج رسالة بها ناتج الجمع مثل الرسالة الموضحة في الصورة التالية.



من الممكن أن تقوم علي سبيل التمرين بإضافة نافذة Static أخري لعرض قيمة الجمع النهائية بها .. بدلا من استخدام اسلوب الرسائل المزعجة.

وفي هذه الحالة ستضع في الاعتبار الحصول علي مقبض تلك النافذة .. مع استخدام دالة SetWindowsText .. وتمرير مقبض النافذة وقيمة الناتج لهذه الدالة.



الفصل السادس

استخدام

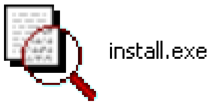
بيئة التطوير المتكاملة

Integrated **D**evelopment **E**nvironment

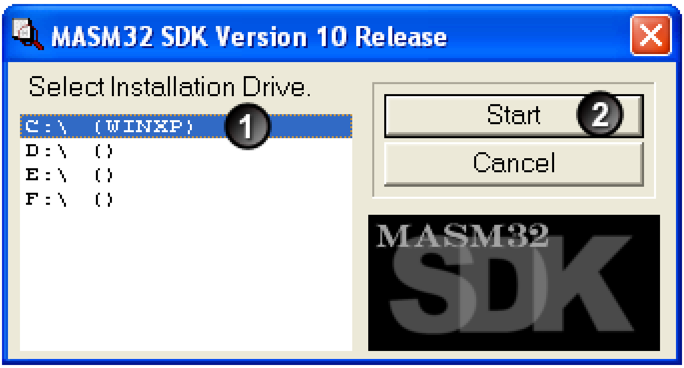
1. تثبيت Masm32.
2. تثبيت وتجهيز RadAsm.
3. استخدام RadAsm في تصميم مشروع بسيط.

أ – خطوات تثبيت MASM32

سنجد الاصدارة v10 مرفقة مع ملفات الكتاب .. فقط نقوم بالضغط علي الملف [install.exe].

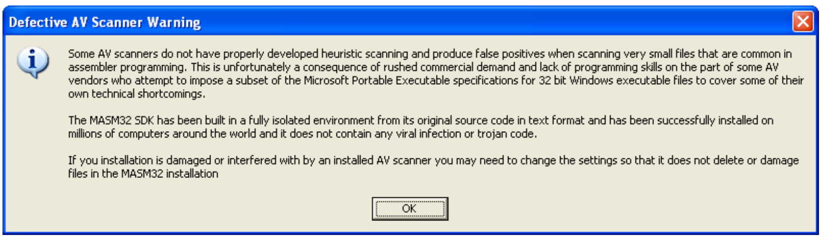


بعد الضغط ستظهر النافذة التالية :

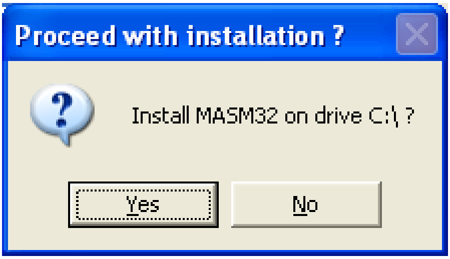


1 - نقوم باختيار مسار التثبيت علي القرصC.
2 - ثم نضغط علي زرStart.

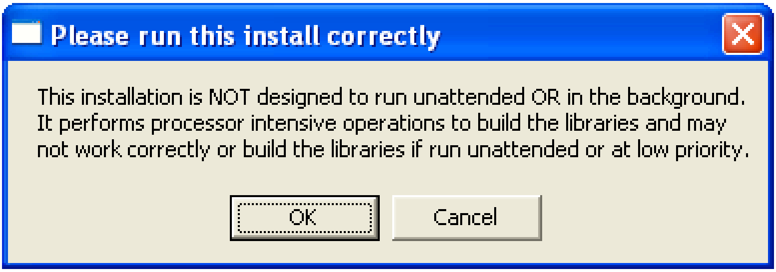
وستظهر نافذة تنبيه من أن بعض برامج فحص الفيروسات تعطي تحذيرات خاطئة لبعض ملفات MASM32 المثبتة وأنها عيب مهاري لمبرمجي تلك البرامج .. فقط قم بالضغط علي الزرOK.



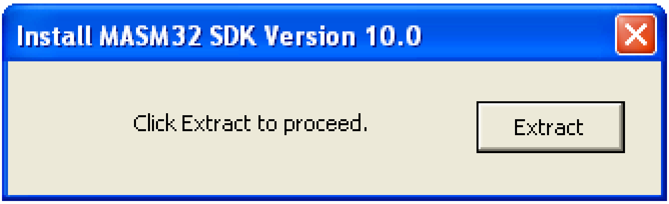
وستظهر نافذة التأكيد التالية .. قم بالتأكيد بالضغط علي زرYes.



بعد الضغط علي yes .. ستظهر رسالة التنبيه التالية:



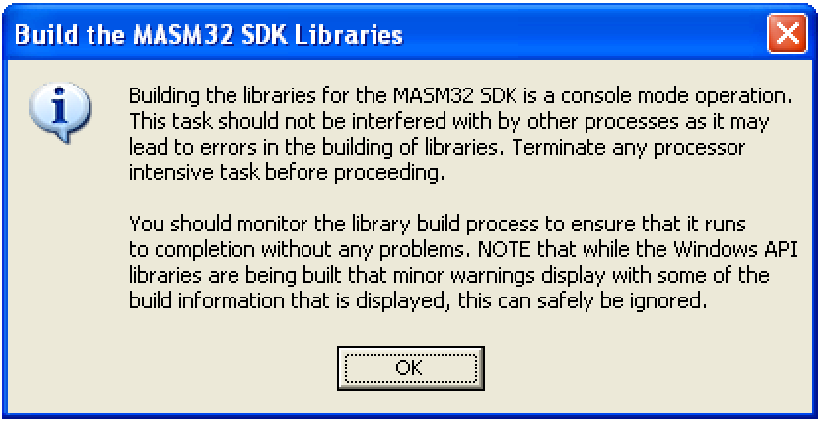
وتفيد بأن يجب إعطاء الأولوية لتثبيتMASM32 .. بحيث لا يعمل معه تطبيق أخر يستهلك موراد الجهاز .. فقط قم بالضغط عليOK .. وسيتم الانتقال في عملية التثبيت للنافذة التالية :



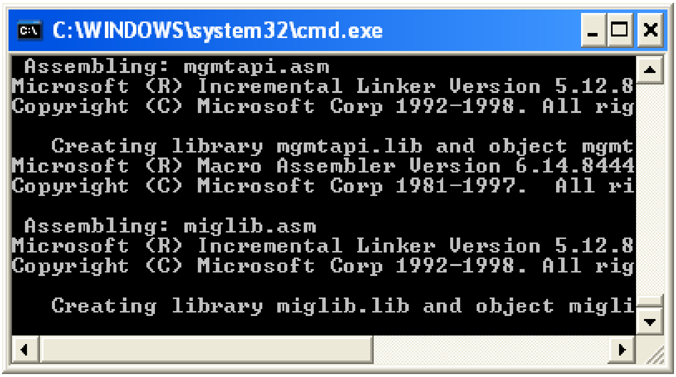
قم بالضغط علي زرExtract .. وستبدأ عملية نسخ الملفات كما هي موضحة في الصورة التالية :



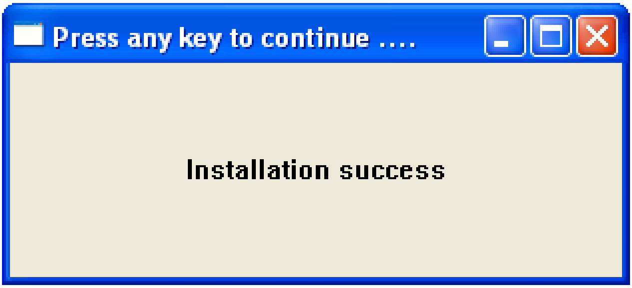
بعد الانتهاء من نسخ الملفات .. ستظهر النافذة التالية :



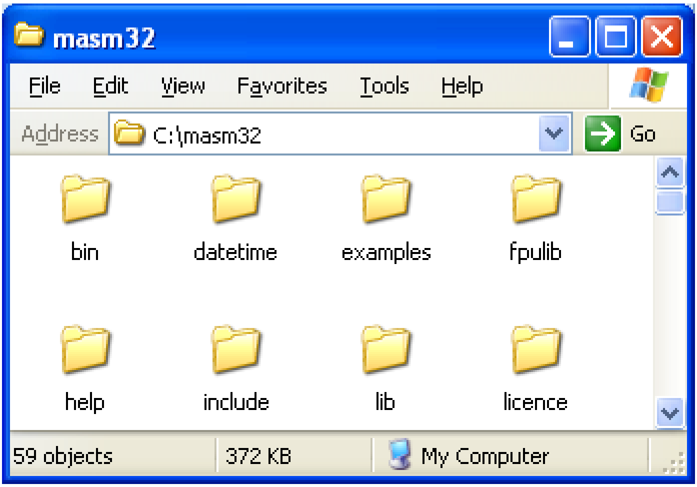
وتفيد بأن إنشاء المكتبات سيكون في مودConsole .. وعند الضغط علي زر OK .. سيستكمل التثبيت إنشاء المكتبات كما في النافذة التالية :



وستأخذ تلك العملية بعض الوقت .. وبعد انتهائها ستظهر نافذة نجاح التثبيت التالية.



ومن المحتمل أنك تريد إلقاء نظره على الأمثلة الرائعة الموجودة بمجلد examples ودراستها والأستفادة منها.



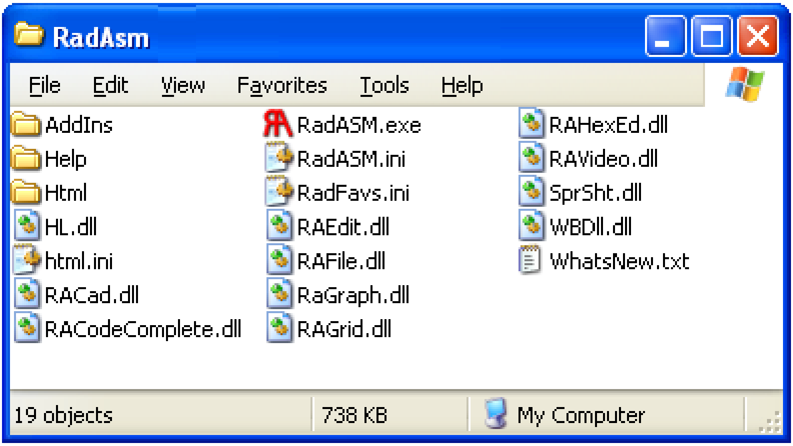
2– تثبيت وتجهيز RadAsm

برنامج RadAsm يعتبر من أفضل برامج IDE للتطوير تطبيقات 32bit باستخدام MASM32.. وسنستعرض فيما يلي كيفية تجهيز هذا البرنامج للعمل.

ستجد نسخة RadASM 2.2.1.4 مرفقة مع ملفات الكتاب ومضاف إليها ملفات الإضافات الخاصة للتطوير بـMASM32 .



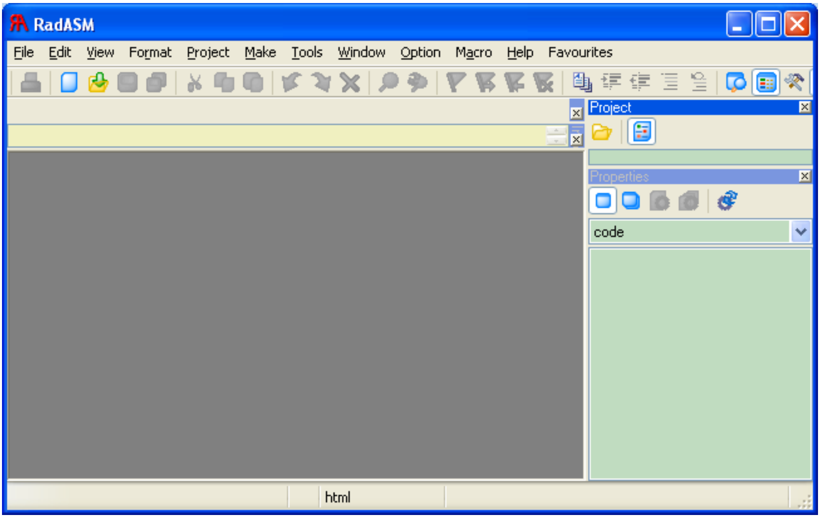
قم بفتح مجلد البرنامج وستجد ملفات البرنامج التالية :



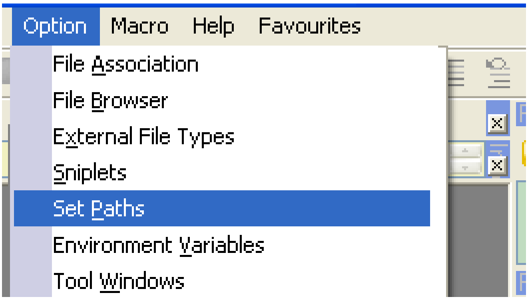
سنقوم بالضغط على ايقونة البرنامج التالية :س



وبعد الضغط سيتم تشغيل البرنامج وستظهر النافذة التالية

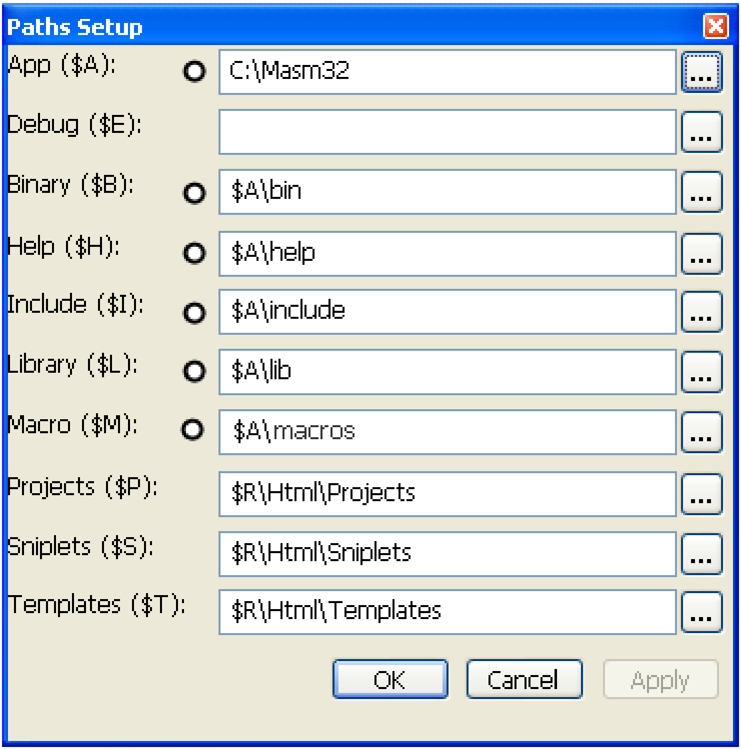


من قائمة Options نختار الأمر Set Paths



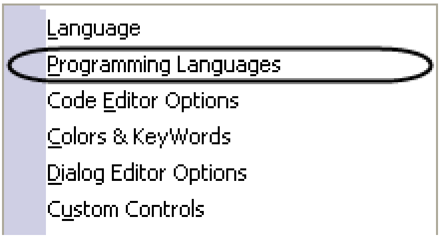
وستظهر نافذة بها حقول مسارات .. نقوم بملء تلك الحقول التي أماما دوائر بالقيم الموضحة في الصورة التالية .. حيث :

$A = "C:\Masm32"
$B = "$A\bin"
$H = "$A\help"
$I = "$A\include"
$L = "$A\lib"
$M = "$A\macros"

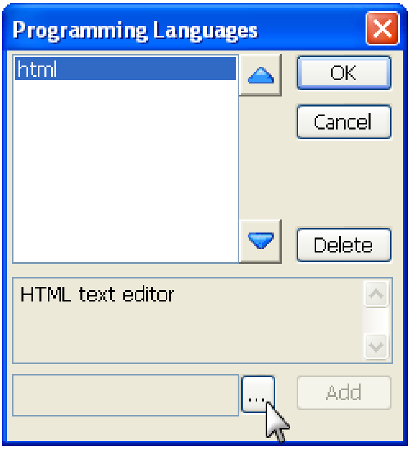


ثم نضغط OK للحفظ ..

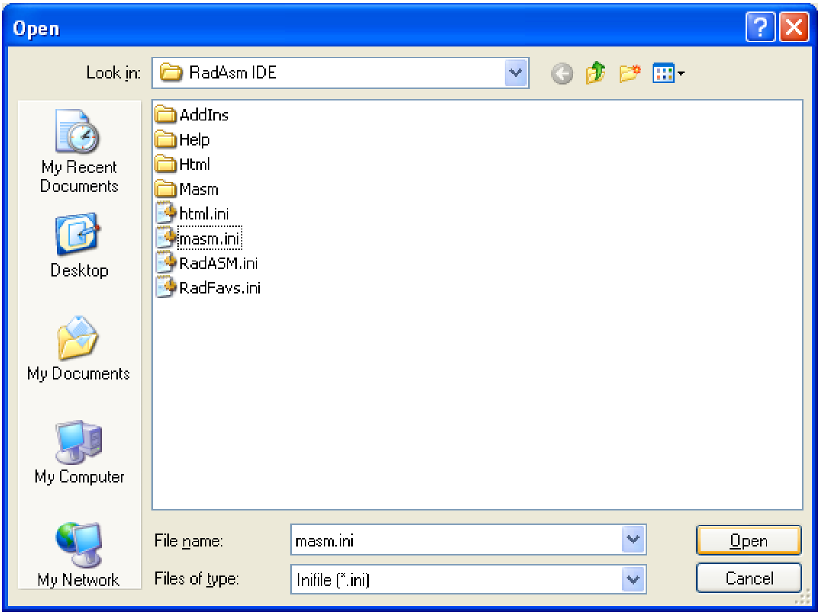
ثم من قائمة Options أيضا نختار أمر لغات البرمجة Programming Langauages.

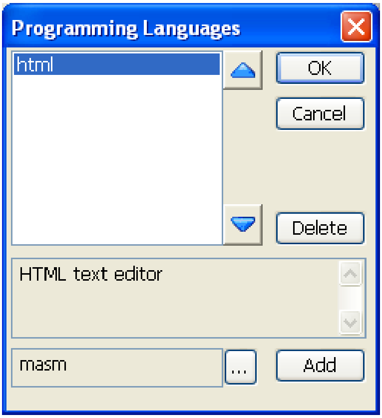


وستظهر النافذة التالية ..

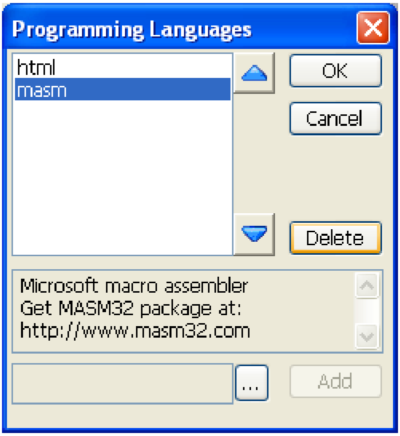


نضغط على زر  لإضافة لغة برمجة جديدة .. ونختار الملف masm.ini





لاحظ ظهور الاسم masm .. ومن ثم نضغط الزر  وستتم إضافة masm إلي لغات البرمجة الحالية التي يدعمها البرنامج.

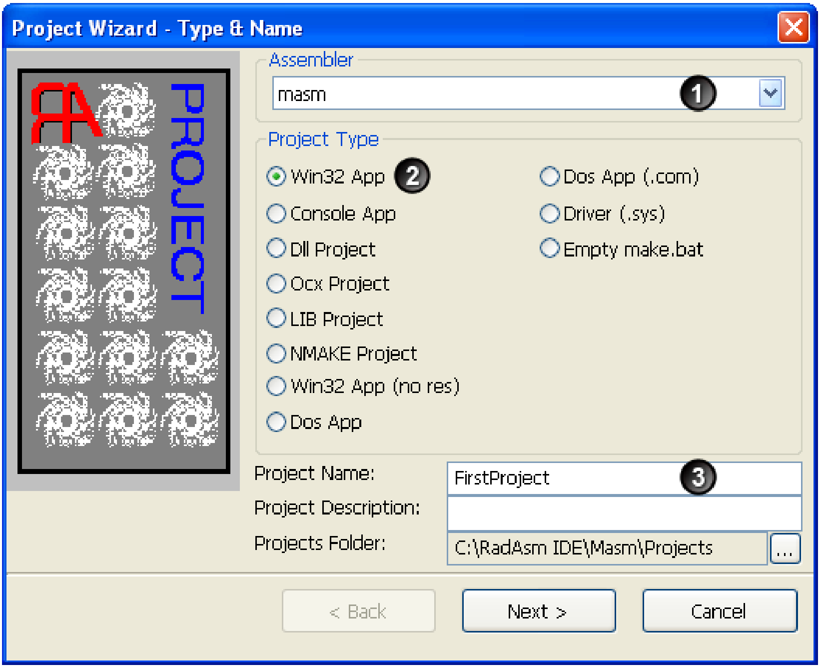


إلي هنا انتهت التغييرات الضرورية .. ومن الممكن أن تريد ضبط سكريبت اللغة وبعض الألوان .. وذلك من قائمة Options واختيار Font Options و Colors & Keywords.

3– إستخدام RadAsm

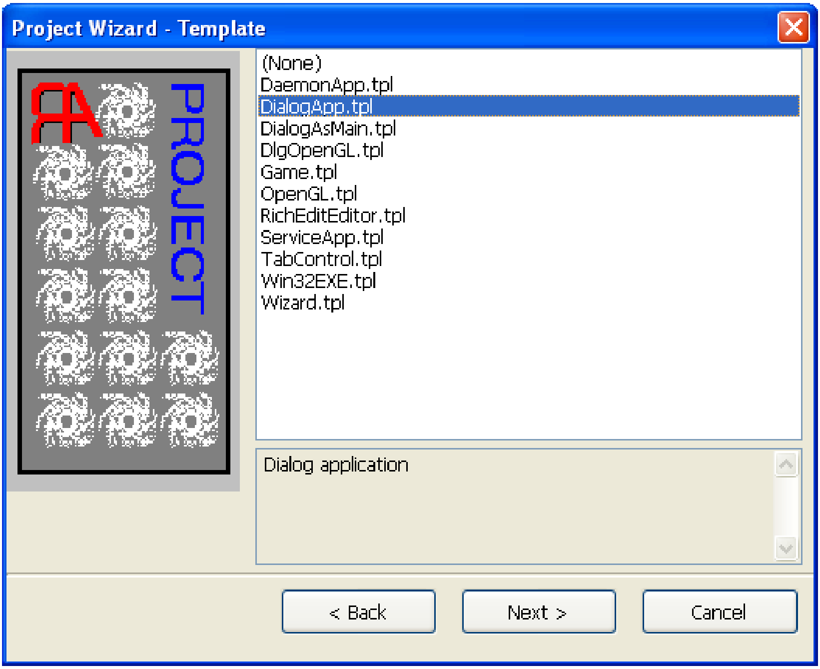
سنقوم الآن بتصميم مشروع بسيط .. يوضح كيفية الاستفاد من بيئة التطوير.

أول خطوة وهي تشغيل RadAsm .. ومن قائمة File نختار New Project وستظهر نافذة الويزارد التالي :



1 - نختار اسمبلر masm من القائمة .
2 - نختار نوع المشروع win32.
3 - نختار اسم للمشروع .. ثم نضغط Next.

الخطوة القادمة بعد الضغط علي Next سيطلب الويزارد اختيار صندوق حواري لإضافته للمشروع .. نختار DialogApp .. وسيقوم الويزارد بإنشاء نافذة حوارية افتراضية .. ومدمج معها الأكواد اللازمة بصورة تلقائية.



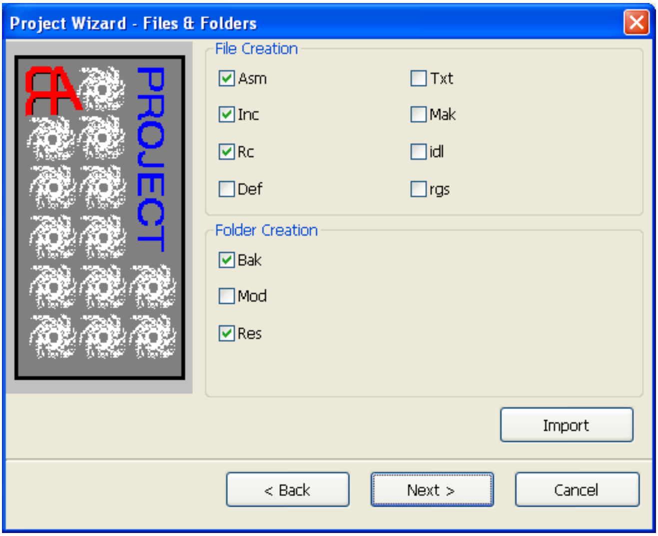
الخطوة التالية بعد الضغط علىNext .. ستظهر نافذة بها تحدد الملفات التي تريد الويزارد أن ينشئها ..

سنختار من صندوق الاختيار العناصر التالية :

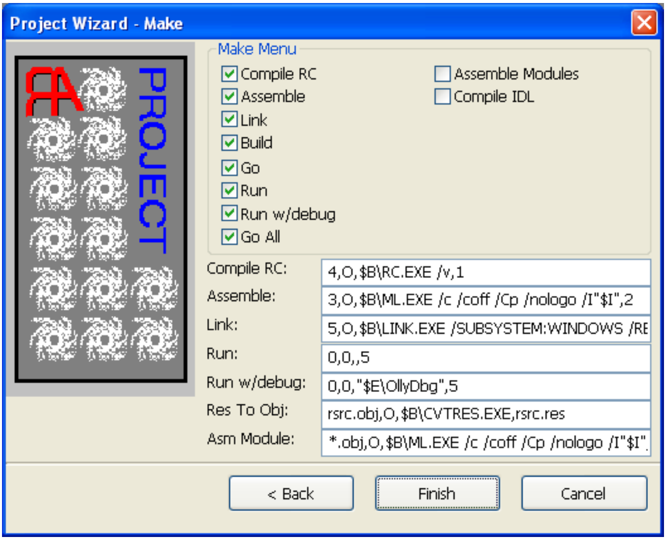
Asm : حيث سيقوم الويزارد بإنشاء ملف الشفرة لنافذة الصندوق الحواري وبه شفرة هذه النافذة.

Inc : سيقوم الويزارد بإدراج كافة المكتبات والتعريفات الخاصة بملف الشفرة ASM في ملف Inc بجوار ملف ASM .. ثم تستخدم تعليمة Include في ملف ASM لتضمين مكتبات وإجراءات وتعريفات المشروع اللازمة.

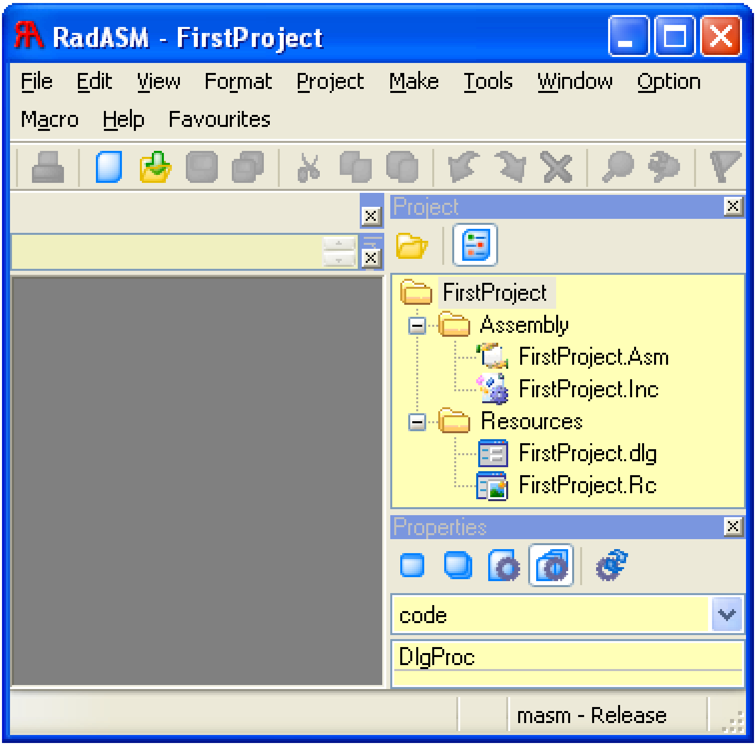
Rc : في حالة إذا أردنا إضافة مصادر Resources للمشروع .. مثل صورة أو ملف صوتي أو نصي.



بعض الضغط علىNext ستظهر النافذة التالية ..



وبها خيارات لخصائص المشروع أثناء التطوير .. نتركها بدون تغيير .. ونضغط Next .. ليتم تنفيذ جميع مراحل الويزارد .. وينشيء لنا المشروع التالي:



لاحظ في نافذة Project وجود ثلاث ملفات مهمه ..

الملف الأول وهو ملف شفرة المشروعFirstProject ..

الملف الثاني وهو ملف التضمين FirstProject.inc ..

الملف الثالث وهو الصندوق الحواري المرئي FirstProject.dlg..

سنلقي نظرة الآن علي محتويات كل من الملفات الثلاث ..

ملف الشفرة FirstProject.asm

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include FirstProject.inc

.code

start:

  invoke GetModuleHandle,NULL
  mov    hInstance,eax

    invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
  invoke ExitProcess,0

;########################################################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

  mov    eax,uMsg
  .if eax==WM_INITDIALOG

  .elseif eax==WM_COMMAND

  .elseif eax==WM_CLOSE
    invoke EndDialog,hWin,0
  .else
    mov    eax,FALSE
    ret
  .endif
  mov    eax,TRUE
  ret

DlgProc endp

end start
```

وسنعود لشرح هذا الملف والدوال الجديدة به

ملف التضمين FirstProject.inc

```

include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib

DlgProc     PROTO  :HWND,:UINT,:WPARAM,:LPARAM

.const

IDD_DIALOG1     equ 101

;#############################################

.data?

hInstance     dd ?

;#############################################
```

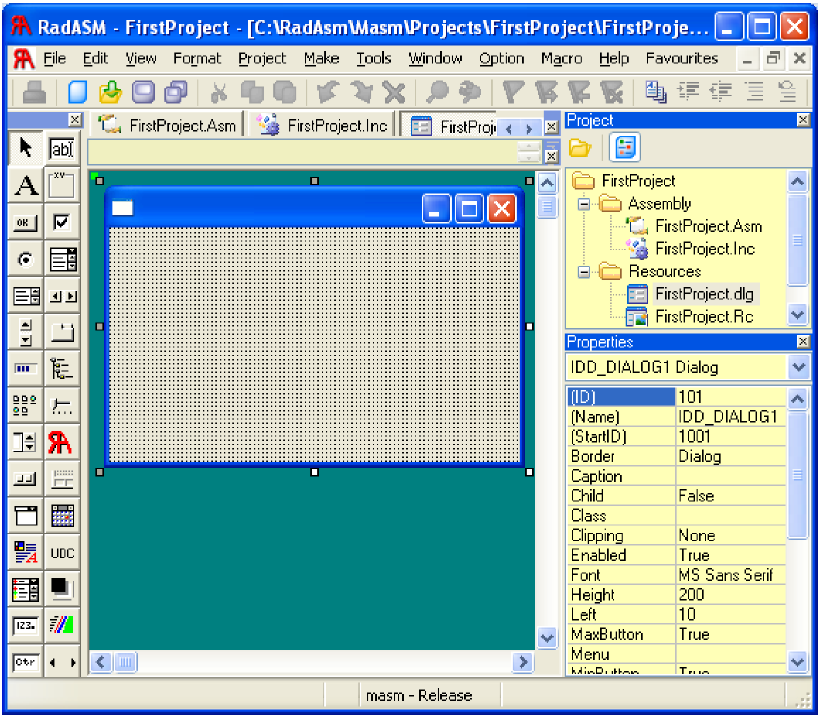
وفي هذا الملف تم تضمين المكتبات التي سيتم استخدامها من قبل ملف الشفرة FirstProject.asm .. وكذلك تعريف إجراء حلقة الرسائل DlgProc.. وكذلك تعريف أقسام البرنامج .

ومن داخل ملف الشفرة نقوم بتضمين هذا الملف عن طريق التعليمة:

include FirstProject.inc

وهي طريقة جيدة لتنظيم محتويات المشروع .. وكذلك إتاحة استخدام محتويات ملف التضمين أكثر من مره بدون إعادة كتابة محتويات الملف في كل ملف شفرة.

ملف الدايلوج FirstProject.dlg

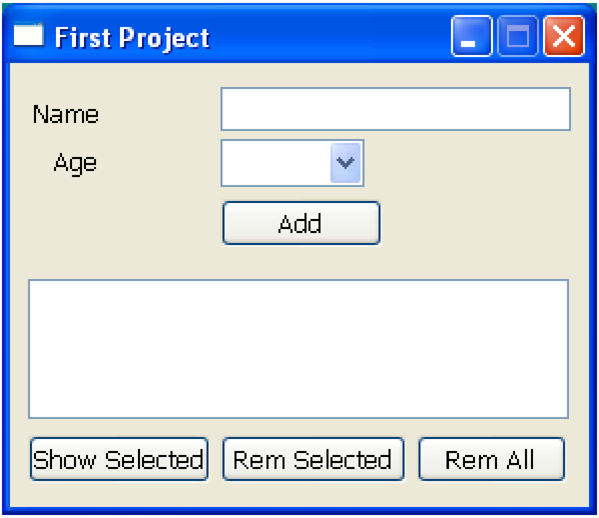


وهو يحتوي علي نموذج التصميم الحواري .. حيث يوجد علي اليسار صندوق الأدوات المعروفة .. وفي المنتصف نافذة النموذج التي سنقوم بالتصميم عليها .. وعلي اليمين توجد نافذة Properties لخصائص الأداة الحالية ..

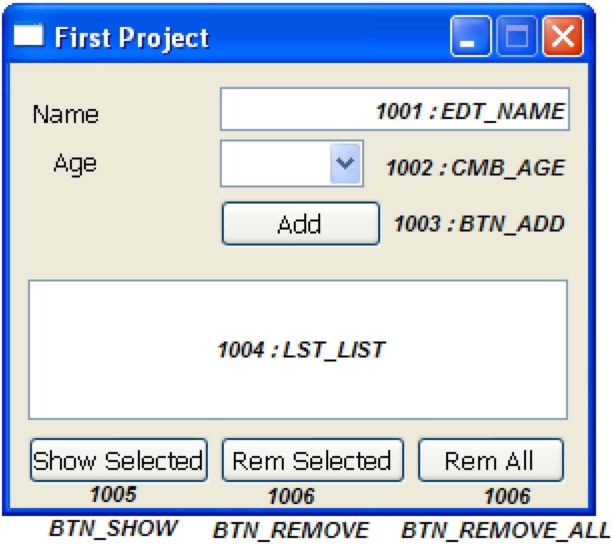
وسنقوم حاليا بتصميم نافذة البرنامج .. تحتوي علي أدوات قياسية معروفة حتي يتسني لنا معرفة كيفية التعامل مع تلك الأدوات ..

Static
EditText
ComboBox
ListBox
Button

ونافذة البرنامج ستبدو على الصورة التالية:

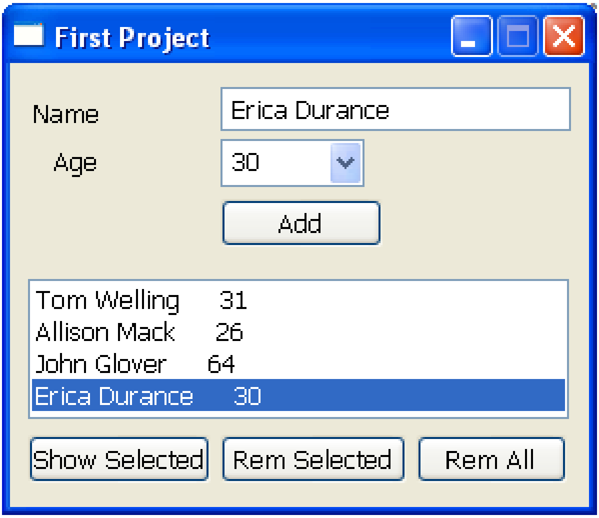


كل نافذة لها معرف ID واسم كما هو موضح في الصورة التالية على الصيغة
ID : Name



الهدف من البرنامج هو أن يقوم المستخدم بكتابة الأسم داخل صندوق تحرير النص TextEdit .. واختيار العمر من صندوق الاختيار ComboBox .. ثم الضغط علي زر Add لإضافة المعلومات إلي القائمة ListBox .. مع وضع زراير للتحكم في محتويات القائمة .. مثل زر إظهار عنصر القائمة المؤشر عليه في رسالة وكذلك زر حذف عنصر القائمة المؤشر عليه .. وأيضا زر حذف جميع عناصر القائمة.

وكمثال أثناء عمل البرنامج .. سيكون الناتج كالتالي :



والآن سنستعرض التغييرات في شفرة كلا من الملفين FirstProjcet.Inc وFirstProject.Asm .. وسنبدأ بشفرة الملف الأول FirstProject.Inc :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc

```

```
includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib

DlgProc     PROTO  :HWND,:UINT,:WPARAM,:LPARAM
NullProc    PROTO

.const
MAX_STRING            equ 20
IDD_DIALOG1           equ 101
EDT_NAME              equ 1001
CMB_AGE               equ 1002
BTN_ADD               equ 1003
LST_LIST              equ 1004
BTN_SHOW              equ 1005
BTN_REMOVE            equ 1006
BTN_REMOVE_ALL        equ 1007

.data
bNull                 db NULL
MSG_Field_Required    db "Missing Field",0
MSG_Project_Name      db "First Project",0
cmbIndex              dd 0
listIndex             dd 0
bName                 db 20 dup(?)
bAge                  db 4  dup(?)
dAge                  dd 0
buffer                db 30  dup(?)
bDelimiter            db 5  dup(32)
;###################################

```

```
41 .data?
42
43 hInstance          dd ?
44 hName              dd ?
45 hCmbAge            dd ?
46 hList              dd ?
47
48 ;###################################
```

الملف كما رأينا احتوي علي كافة المكتبات والمتغيرات وتعريف الإجراءات التي سنستخدمها في البر

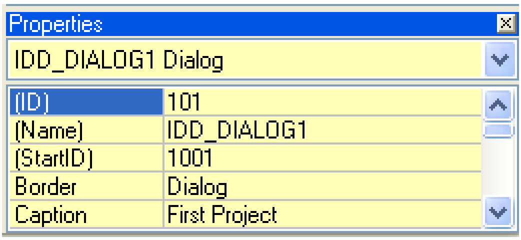
الجديد هو تضمين مكتبة Comctl32 أو مكتبة الـ Common Controls التي تحتوي علي جميع الأدوات القياسية التي سنقوم باستخدامها في المشروع..

السطر 14 : نقوم بتعريف إجراء النافذة Window Procedure للنافذة الحوارية Dialog .. والتي سيأتي الحديث عنها بعد قليل.

السطر 15 : نقوم بتعريف إجراء خاص بحذف قيمة المتغيرات.

السطر 18 : نقوم بتعريف ثابت سنستخدمه فيما بعد لتحديد أقصي حجم لقيمة يتم نسخها.

السطر 19 : ويتم إنشائه تلقائيا من قبل RadAsm عند اختيار التمبلت DialogAPP من الويزارد وفيه يحدد معرف النموذج الحواري Dialog .



يمكنك تغييره لاي اسم تريد .. لكن لا تنسي تغيير الثابت ايضا.

السطور 20-26 : نقوم بالإعلان عن معرفات الأدوات التي سنستخدمها داخل الشفرة.

السطر 29 : حجز متغير بايت ليحمل القيمةNULL.

السطر 30 : حجز متغير بايت ليحمل سلسلة نصية بها رسالة تحذير .. إذا ترك المستخدم حقل الاسم فارغ عند الضغط علي زرAdd.

السطر 31 : حجز متغير ليحمل اسم المشروع .. يفيد عند عرض عنوان الرسالة .

السطر 32 : حجز متغير عددي ليحمل مؤشر العنصر داخل صندوق الاختيار ComboBox.

السطر 33 : حجز متغير ليحمل مؤشر العنصر داخل صندوق القائمة ListBox.

السطر 34 : حجز 20 بايت لمتغير ليحمل الاسم الذي سيدخله المستخدم..حيث التعليمة DUP تأخذ الصيغة التالية:

Number DUP(value)

حيث Number هو عدد البايتات .. و Value هي القيمة الابتدائية التي ستحتويها تلك البايتات وعند تمرير(?) فهذا يعني انه لا توجد قيمة ابتدائية.

السطر 35 : حجز 4 بايتات لتخزين ادخال المستخدم للعمر.

السطر 36 : حجز متغير عددي لتخزين القيمة العددية للعمر من السطر35.

السطر 37 : حجز مخزن بحجم 30 بايت .. لاستخدامه في عمليات التخزين المؤقت وكذلك ليحمل سلسلة البيانات النهائية الاسم + الفاصل + العمر.

السطر 38 : حجز متغير ليمثل الفاصل بين الاسم والعمر .. وتم استخدام فاصلة المسافة space حيث يقابل المسافة في جدول الآسكي القيمة 32 وتم تكرارها 5 مرات.

السطر 44 : متغير عددي سيحمل مقبض نافذة صندوق الإدخال النصي للاسم EDT_NAME.

السطر 45 : متغير سيحمل مقبض نافذة صندوق الاختيارCMB_AGE.

السطر 46 : متغير عددي سيحمل مقبض نافذة صندوق القائمةLST_LIST.

الآن نستعرض ملف الشفرةFirstProject.Asm :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include FirstProject.inc

.code

start:

 invoke GetModuleHandle,NULL
 mov    hInstance,eax

 invoke InitCommonControls
 invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
 invoke ExitProcess,0

;########################################################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

 mov    eax,uMsg
 .if eax==WM_INITDIALOG
    invoke GetDlgItem,hWin,EDT_NAME
    mov    hName,eax
    invoke GetDlgItem,hWin,CMB_AGE
    mov    hCmbAge,eax
    invoke GetDlgItem,hWin,LST_LIST
    mov    hList,eax

```

```
    mov cmbIndex , 15
    .Repeat
      invoke dwtoa ,cmbIndex,ADDR buffer
      ;invoke SendMessage,hCmbAge,CB_INSERTSTRING,-1,addr buffer
      invoke SendDlgItemMessageA,hWin,CMB_AGE,CB_INSERTSTRING,-1,\
                                        addr buffer
      Inc cmbIndex
    .Until cmbIndex == 36

  .elseif eax==WM_COMMAND
    .IF lParam != 0
        mov eax,wParam
        mov bx,ax
        shr eax,16
       .if ax==BN_CLICKED
        .if bx==BTN_ADD
           invoke NullProc
           invoke GetDlgItemText,hWin,EDT_NAME,addr bName,MAX_STRING
           invoke lstrcmp, addr bName,addr bNull
           je RequiredFileds
           invoke GetDlgItemText,hWin,CMB_AGE,addr bAge,MAX_STRING
           invoke atol,ADDR bAge
           mov dAge,eax
           invoke dwtoa ,dAge,ADDR bAge
           invoke lstrcat,addr buffer,addr bName
           invoke lstrcat,addr buffer,addr bDelimiter
           invoke lstrcat,addr buffer,addr bAge

          invoke SendDlgItemMessageA,hWin,LST_LIST,LB_INSERTSTRING,-1,\
                                        addr buffer
          ret
      RequiredFileds:
          invoke MessageBox,NULL,addr MSG_Field_Required,NULL,MB_OK
          invoke SetFocus,hName

       .elseif bx==BTN_SHOW
           invoke SendMessage,hList,LB_GETCURSEL,0,0
           mov listIndex, eax
           invoke SendMessage,hList,LB_GETTEXT,listIndex,ADDR buffer
           invoke MessageBox,NULL,addr buffer,ADDR MSG_Project_Name,\
                        MB_OK
```

```
71
72          .elseif bx==BTN_REMOVE
73            invoke SendMessage,hList,LB_GETCURSEL,0,0
74            mov listIndex, eax
75            invoke SendMessage,hList,LB_DELETESTRING,listIndex,0
76
77          .elseif bx==BTN_REMOVE_ALL
78            invoke SendMessage,hList,LB_GETCOUNT,0,0
79             mov listIndex, eax
80
81             .Repeat
82                     dec listIndex
83                     invoke SendMessage,hList,LB_DELETESTRING,listIndex,0
84             .Until listIndex==0
85          .ENDIF
86        .ENDIF
87      .ENDIF
88   .elseif eax==WM_CLOSE
89    invoke EndDialog,hWin,0
90   .else
91    mov   eax,FALSE
92    ret
93   .endif
94   mov   eax,TRUE
95   ret
96
97 DlgProc endp
98
99 NullProc proc
100  invoke lstrcpy,addr buffer,addr bNull
101  invoke lstrcpy,addr bName,addr bNull
102  invoke lstrcpy,addr bAge,addr bNull
103  ret
104 NullProc endp
105 end start
106
```

السطر 5 : نقوم بتضمين الملف FirstProject.Inc.

السطر 14 : يتم استدعاء دالة InitCommonControls .. وهي من دوال مكتبة Comctl32.dll .. ووظيفتها ضمان تحميل تلك المكتبة .. ولا يوجد لها معاملات ولا تعود بقيمة.

The **InitCommonControls** function ensures that the common control dynamic-link library (DLL) is loaded.

```
void InitCommonControls(VOID);
```

السطر 15 : يتم استدعاء دالة DialogBoxParam ووظيفتها إنشاء صندوق حواري من قالب Template بشكل يسهل التعامل معه بصورة أفضل من تسجيل وإنشاء نافذة ومراقبة احداثها كما فعلنا في الفصل الرابع.

ويوجد نوعان من الصناديق الحوارية :

النوع الأول Modal ويتم إنشائه بواسطة دالة DialogBoxParam .. وهو يجبرك علي التعامل مع النافذة الحوارية أولا قبل الانتقال إلي نافذة أخري حوارية في نفس التطبيق..

النوع الثاني Modeless ويتم إنشائه بواسطة دالة CreateDialogParam وهو يجعلك تتعامل مع أي نافذة حوارية في التطبيق في أي وقت.

The **DialogBoxParam** function creates a modal dialog box from a dialog box template resource. Before displaying the dialog box, the function passes an application-defined value to the dialog box procedure as the *lParam* parameter of the WM_INITDIALOG message. An application can use this value to initialize dialog box controls.

```
int DialogBoxParam(
  HINSTANCE hInstance,         // handle to application instance
  LPCTSTR lpTemplateName,      // identifies dialog box template
  HWND hWndParent,             // handle to owner window
  DLGPROC lpDialogFunc,        // pointer to dialog box procedure
  LPARAM dwInitParam           // initialization value
);
```

The **CreateDialogParam** function creates a modeless dialog box from a dialog box template resource. Before displaying the dialog box, the function passes an application-defined value to the dialog box procedure as the *lParam* parameter of the the WM_INITDIALOG message. An application can use this value to initialize dialog box controls.

```
HWND CreateDialogParam(
  HINSTANCE hInstance,        // handle to application instance
  LPCTSTR lpTemplateName,     // identifies dialog box template
  HWND hWndParent,            // handle to owner window
  DLGPROC lpDialogFunc,       // pointer to dialog box procedure
  LPARAM dwInitParam          // initialization value
);
```

افتراضيا يقوم RadAsm بانشاء النوع Modal .. وبالتالي إذا أردت التعامل مع أكثر من نافذة حوارية في نفس الوقت فلا تنسى تغيير الدالة.

وكلا الدالتين لهما نفس المعاملات التالية :

1 - HINSTANCE
مقبض التطبيق والذي حصلنا عليه بواسطةGetModuleHandle.

2 - LPCTSTR
معرف قالب الصندوق الحواري .. وهذا المعامل إما أن يأخذ عنوان لنص به اسم لقالب جاهز .. أو أن يأخذ قيمة تشير لمعرف مصدر قالب الصدوق الحواري .. وفي حالتنا هنا .. يوجد ملف مصدر Resource وهو FirstProjectDlg.Rc والذي يصير بعد ترجمته FirstProject.RES به قالب الصندوق الحواري .. فقط يتم تمرير المعرف الخاص به لهذا المعامل.

3 - HWND
يحدد هذا المعامل النافذة الأصل لهذا الصندوق الحواري.. وبما أنها نافذة واحدة..فيتم تمرير القيمة NULL.

4 - DLGPROC
مؤشر لإجراء الصندوق الصندوق الحواري والذي سيتم معالجة الرسائل به.. وإجراء النافذة هنا هوDlgProc.

5 - LPARAM
معامل إضافي .. في حالة إذا أردت تمرير قيمة ابتدائية في المعامل lParam لإجراء الصندوق لرسالة WM_INITDIALOG والتي يستقبلها الإجراء عند إنشاء نافذة الصندوق الحواري..وقبل ظهور نافذة الصندوق مباشرة.

ومن الداخل فإن كلا من الدالتين يستخدمان دالة CreateWidnowEx لإنشاء نافذة الصندوق الحواري Dialog Box.

السطر 20 : بداية إجراء نافذة الصندوق الحواري.

السطر 23 : يتم تحليل رقم الرسالة .. فإذا كانت WM_INITDIALOG فهذا يعني أنه تم إنشاء الصندوق الحواري.

السطور 24- 29 : نقوم بمعرفة مقابض نوافذ كل من صندوق الادخال والاختيار والقائمة.. عن طريق دالة GetDlgItem ..

The **GetDlgItem** function retrieves the handle of a control in the specified dialog box.

```
HWND GetDlgItem(
  HWND hDlg,          // handle of dialog box
  int nIDDlgItem      // identifier of control
 );
```

حيث في المعامل الأول نقوم بتمرير مقبض نافذة الصندوق الحواري المحتوي علي نافذة أداة التحكم وهو ممرر مع إجراء الصندوق .. بينما في المعامل الثاني نمرر معرف الأداة التي نريد الحصول علي مقبضها.. وتعود الدالة بقيمة المقبض في المسجل eax ونتقله للمتغيرات المخصصة لذلك لاستخدامه فيما بعد.

السطر 31 : نقوم بنسخ القيمة الابتدائية 15 إلي المتغير cmbIndex .. وهو المتغير الذي سيستخدم لملء صندوق الاختيار ComboBox الخاص بالعمر بالأعداد من 15 إلي 35.

السطر 32 : نبدأ بداية حلقة تكرار بواسطة التعليمة Repeat.

السطر 33 : نقوم بتحويل القيمة العددية إلي قيمة نصية تمهيدا لإضافتها في صندوق الاختيار ComboBox .. ونستخدم لذلك المخزن المؤقت buffer مع استخدام دالة التحويل [dwtoa] DWORD TO ASCII .. وهي من دوال مكتبة MASM32 .. مع تمرير القيمة المراد تحويلها في المعامل الأول,وعنوان المتغير النصي الذي سيتم التحويل إليه في المعامل الثاني.

السطر 34- 35 : كلا السطرين يعملان علي إضافة عنصر إلي صندوق الاختيار .. حيث توجد طريقتين لعمل هذه الإضافة ..

- إما استخدام دالة إرسال الرسائل SendMessage
- أو استخدام الدالة الخاصة بصندوق الحوار SendDlgItemMessage

The **SendMessage** function sends the specified message to a window or windows. The function calls the window procedure for the specified window and does not return until the window procedure has processed the message. The **PostMessage** function, in contrast, posts a message to a thread's message queue and returns immediately.

```
LRESULT SendMessage(
  HWND hWnd,          // handle of destination window
  UINT Msg,           // message to send
  WPARAM wParam,      // first message parameter
  LPARAM lParam       // second message parameter
 );
```

دالة SendMessage تعتبر دالة شاملة .. وبها أربعه معاملات قد أصبحوا معروفين لك الآن ..
الأول وهو مقبض النافذة المراد إرسال الرسالة لها ..
الثاني وهو قيمة الرسالة..
الثالث والرابع هما البيانات الإضافية للرسالة ..

والرسالة التي نريد إرسالها لنافذة صندوق الاختيار ComboBox هي رسالة إضافة نص CB_ADDSTRING أو CB_INSERTSTRING..

المعامل wParam يحدد هنا موضع العنصر الذي تريد إضافته في القائمة .. ونمرر القيمة (1-) لتركه لنظام الموضع الافتراضي..وهو الإضافة المتتابعه.

المعامل lParam يحدد هنا النص المراد إضافته .. وبالتالي نمرر عنوان المتغير buffer.

```
CB_SETEDITSEL         equ 142h
CB_ADDSTRING          equ 143h
CB_DELETESTRING       equ 144h
CB_DIR                equ 145h
CB_GETCOUNT           equ 146h
CB_GETCURSEL          equ 147h
CB_GETLBTEXT          equ 148h
CB_GETLBTEXTLEN       equ 149h
CB_INSERTSTRING       equ 14Ah
CB_RESETCONTENT       equ 14Bh
CB_FINDSTRING         equ 14Ch
```

وبالنسبة لدالة SendDlgItemMessage فهي لا تختلف عن سابقتها سوي أنها مخصصة لنوافذ التحكم الخاصة بالصناديق الحوارية فقط .. حيث تم إضافة معامل إضافي لها وهو المعامل الثاني IDDlgItem ونمرر به معرف أداة التحكم داخل البرنامج.. بينما المعامل الأول يمثل مقبض نافذة الصندوق الحواري نفسه وليس مقبض نافذة أداة التحكم.

The **SendDlgItemMessage** function sends a message to the specified control in a dialog box.

```
LONG SendDlgItemMessage(
  HWND hDlg,          // handle of dialog box
  int nIDDlgItem,     // identifier of control
  UINT Msg,           // message to send
  WPARAM wParam,      // first message parameter
  LPARAM lParam       // second message parameter
);
```

السطر 37 : نقوم بإضافة واحد للعداد cmbIndex بتعليمة Inc.

السطر 38 : شرط التكرار هنا هو أن يستمر التكرار حتي تصل قيمة العداد cmbIndex إلي القيمة 36.

السطر 46 : نبدأ بمعالجة رسالة الضغط لأزرار النافذة ..

السطر 47 : نستدعي الإجراء NullProc والذي يقوم بحذف قيم المتغيرات المستخدمه في تخزين البيانات.

السطر 48 : هل تتذكر دالة GetWindowText من الفصل الخامس .. يمكن استخدامها بالطبع للحصول علي الاسم من صندوق إدخال الاسم .. ولكن توجد دالة جديدة من دوال التعامل مع الصناديق الحوارية وهي دالة GetDlgItemText

The **GetDlgItemText** function retrieves the title or text associated with a control in a dialog box.

```
UINT GetDlgItemText(
  HWND hDlg,            // handle of dialog box
  int nIDDlgItem,       // identifier of control
  LPTSTR lpString,      // address of buffer for text
  int nMaxCount         // maximum size of string
 );
```

حيث :
المعامل الأول وهو مقبض الصندوق الحواري.
المعامل الثاني وهو معرف أداة التحكم.
المعامل الثالث وهو عنوان المتغير الذي سيتم وضع النص به.
المعامل الرابع وهو أكبر حجم للنص الذي سيتم نسخه.

السطر 49 : نقوم هنا بمقارنة النص المنسوخ من صندوق الإدخال للاسم بالقيمة NULL .. وذلك تحسبا أن يكون المستخدم لم يقم بكتابة الاسم .. وبالتالي نظهر له رسالة تفيد ذلك .. وتم استخدام الدالة lstrcmp لعمل المقارنة ..

The **lstrcmp** function compares two character strings. The comparison is case sensitive.

```
int lstrcmp(
  LPCTSTR lpString1,    // address of first string
  LPCTSTR lpString2     // address of second string
 );
```

السطر 50 : نفحص حالة المقارنة بواسطة تعليمة القفز je .. فإذا كانت قيمة الاسم تساوي NULL فيتم القفز لرسالة التحذير بالسطر62.

السطر 51 : إذا وصل مسار التنفيذ لهنا فهذا يعني أنه تم وضع اسم بصندوق الإدخال ..وبالتالي نستكمل التنفيذ ونقوم بإتيان العمر بصندوق الاختيار.

السطر 52 : نقوم بتحويل القيمة النصية للعمر إلي قيمة عددية بواسطة دالة [atol] Ascii To Long .. حيث نمرر لها عنوان المتغير النصي .. وتعود بقيمة النص في المسجلeax.

السطر 53 : نقوم بنسخ قيمة المسجل إلي المتغير العددي للعمر.

السطر 54 : نقوم بتخزين القيمة العددية للعمر إلي متغير نصي مره أخري.

السطور 55-57 : نقوم ببناء النص النهائي في المخزنbuffer .. حيث نقوم بتجميع النصوص بواسطة الدالةlstrcat ..

The **lstrcat** function appends one string to another.

```
LPTSTR lstrcat(
  LPTSTR lpString1,   // address of buffer for concatenated strings
  LPCTSTR lpString2   // address of string to add to string1
);
```

حيث المعامل الأول وهو عنوان المتغير المخزن للنص النهائي .. بينما المعامل الثاني هو عنوان المتغير النصي المراد إضافته لهذا المخزن.

السطر 59 : نقوم باستخدام دالة SendDlgItemMessage مرة أخري لإضافة النص النهائي في المخزن buffer إلي القائمة ListBox ولكن هنا سنستخدم رسالة الإضافة LB_INSERTSTRING.

السطر 61 : تعليمة ret مهمه هنا لكي لا ينتقل مسار التنفيذ إلي السطر رقم 62 وتظهر رسالة التحذير.

السطر 64 : بعد ظهور رسالة التحذير .. من الأفضل أن نعطي التنشيط الحالي لصندوق الإدخال النصي ..ونستخدم ذلك دالة SetFocus .. والمعامل الوحيد لها وهو مقبض النافذة التي تريد جعلها نشطه.

The **SetFocus** function sets the keyboard focus to the specified window. The window must be associated with the calling thread's message queue.

```
HWND SetFocus(
  HWND hWnd        // handle of window to receive focus
 );
```

السطر 66 : بداية شفرة زر عرض العنصر المنشط في القائمة.

السطر 67 : نقوم بإرسال رسالة LB_GETCURSEL لمعرفة موضع العنصر المنشط في القائمة.. ولك مطلق الحرية في استخدام الدالة التي تريدها لإرسال تلك الرسالة..

السطر 68 : نسخ موضع العنصر المنشط من المسجلeax .

السطر 69 : نقوم بإرسال رسالة إتيان النص من القائمة LB_GETTEXT مع تمرير موضع العنصر المنشط listIndex والذي حصلنا عليه في المعامل wParam .. وأيضا تمرير عنوان المخزن الذي سيتم وضع النص به في المعامل lParam.

السطر 70 : نظهر النص الذي حصلنا عليه من القائمة في رسالة.

السطر 72 : بداية شفرة زر حذف العنصر المنشط في القائمة.

السطر 73 : نقوم أيضا بمعرفة موضع العنصر المنشط في القائمة.

السطر 75 : ثم نرسل للقائمة رسالة حذف النص LB_DELETESTRING مع تمرير موضع العنصر المراد حذفه.

السطر 77 : بداية شفرة حذف جميع العناصر في القائمة.

السطر 78 : نقوم بإتيان عدد العناصر الموجودة في القائمة وذلك بإرسال رسالة LB_GETCOUNT.

السطر 79 : نقوم بنسخ عدد العناصر إلي المتغيرlistIndex.

السطور 81 – 84 : نبدأ حلقة تكرار شرط استمرارها هو أن تكون قيمة المتغير listIndex لا تساوي الصفر .. ونقوم بتقليل قيمة هذا المتغير كل مره .. ثم نرسل رسالة حذف للنص إلي النافذة.

لاحظ أنه إذا كانت عدد العناصر التي حصلنا عليها هو 4 .. فهذا يعني أن العناصر مؤشرة من الصفر إلي ثلاثة .. وبالتالي نقوم بتقليل قيمة المتغير أولا ثم نقوم بالحذف.

السطر 89 : عند استقبال رسالة الإغلاق WM_Close .. فنقوم بإنهاء الصندوق الحواري بواسطة الدالة EndDialog مع تمرير مقبض الصندوق في المعامل الأول .. والقيمة صفر كقيمة لكود الإنهاء.

The **EndDialog** function destroys a modal dialog box, causing the system to end any processing for the dialog box.

```
BOOL EndDialog(
  HWND hDlg,        // handle to dialog box
  int nResult       // value to return
);
```

السطور 91 – 95 : إذا تم معالجة الرسالة فنقوم بإرجاع القيمة True, بينما إذا لم تتم معالجة الرسالة فنقوم بإرجاع القيمة False ليقوم إجراء النافذة الافتراضي بمعالجتها.

السطور 99 – 104 : نقوم بحذف قيم بعض المتغيرات .. عن طريقة نسخ القيمة NULL إليها بواسطة الدالة lstrcpy ..

The **lstrcpy** function copies a string to a buffer.

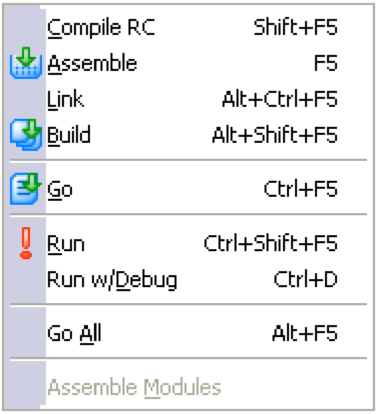
```
LPTSTR lstrcpy(
  LPTSTR lpString1,         // address of buffer
  LPCTSTR lpString2         // address of string to copy
);
```

حيث :
المعامل الأول هو عنوان المتغير المراد نسخ القيمة إليه.

المعامل الثاني وهو عنوان المتغير المراد نسخ القيمة منه.
في النهاية نقوم بترجمة البرنامج من قائمة Make ونختار Compile RC .. أو نضغط علي Shift + F5 .. وذلك لتكوين ملف المصدر للصندوق الحواري

ثم نختار Build من نفس القائمة أو بالضغط علي Alt+Shift+F5

ثم نقوم بتشغيل البرنامج باختيار Run من نفس القائمة أو بالضغط علي Ctrl+Shift+F5





الفصل السابع

التعامل مع الملفات

Working With Files

1. دوال الملفات.
2. تصميم البرنامج.
3. شرح شفرة البرنامج.

1 – دوال التعامل مع الملفات

سنتعلم في هذا الفصل .. كيفية قراءة وكتابة الملفات النصية .. وسنتناول شرح تفصيلي للدوال التي سنستخدمها أولا .. ثم سنضع تلك الدوال موضع التنفيذ في برنامج عملي.

❖ **<u>دالة CreateFile</u> :**

تستخدم هذه الدالة في إنشاء وفتح الملفات .. بالإضافة إلي أنه يمكن استخدامها لأشياء أخري.

```
The CreateFile function creates or opens the following objects and
 returns a handle that can be used to access the object:

· files
· pipes
· mailslots
· communications resources
· disk devices (Windows NT only)
· consoles
· directories (open only)

HANDLE CreateFile(
LPCTSTR lpFileName,                    // pointer to name of the file
DWORD dwDesiredAccess,                 // access (read-write) mode
DWORD dwShareMode,                     // share mode
LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                                       // pointer to security attributes
DWORD dwCreationDistribution,          // how to create
DWORD dwFlagsAndAttributes,            // file attributes
HANDLE hTemplateFile                   // handle to file with attributes to copy
);
```

بعد استدعاء الدالة بنجاح تعود بمقبض للملف .. أو تعود بالقيمة صفر والتي تشير لفشل الدالة .. وللدالة 7 معاملات Parameters هما كالأتي:

1 - lpFileName :

مؤشر لاسم الملف المراد فتحه أو إنشاءه.. ولاحظ أن اسم الملف يجب أن يكون طوله أقصاه 256 حرف .. وإذا أردت استخدام طول أكبر من الطول المحدد فيجب استدعاء النسخة الـ Wide من الدالة وهي "CreateFileW" حيث الاستدعاء الافتراضي يكون للنسخة "CreateFileA".

2 - dwDesiredAccess:

يحدد نوع العملية المراد تنفيذها علي الملف .. قراءة أو كتابة.
وياخذ أي من القيمتين التاليتين:

- GENERIC_READ : فتح الملف للقراءة.
- GENERIC_WRITE : فتح الملف للكتابة.

3 - dwShareMode

يحدد العمليات التي تستطيع التطبيقات الأخري المشاركة بإجرائها علي الملف .. ويأخذ أي من القيم التالية:

- FILE_SHARE_DELETE : الموافقة علي حذف الملف.
- FILE_SHARE_READ : الموافقة علي قراءة الملف.
- FILE_SHARE_WRITE: الموافقة علي الكتابة في الملف.

أو يمكنك وضعه بصفر لكي لا تسمح بمشاركة الملف مع أي تطبيق أخر.

4 - lpSecurityAttributes

مؤشر لبنية SECURITY_ATTRIBUTES تحدد المقبض يمكن توريثه للتطبيقات المشتقة أم لا.. ولن نستخدمه ونضع قيمتهNULL.

5 - dwCreationDistribution

يحدد العملية التي ستتم علي الملف في حالة وجوده والعملية التي ستتم علي الملف في حالة عدم وجوده .. ويأخذ أي من القيم التالية:

القيمة	المعني
CREATE_NEW	إنشاء ملف جديد .. وتفشل الدالة في حالة وجود الملف من قبل
CREATE_ALWAYS	إنشاء الملف دائما .. والكتابة علي

	الملف إذا كان موجود من قبل
OPEN_EXISTING	فتح ملف موجود مسبقا.. وتفشل الدالة إذا كان الملف غير موجود
OPEN_ALWAYS	فتح الملف موجود مسبقا .. وإذا لم يكن فستقوم الدالة بإنشائه
TRUNCATE_EXISTING	فتح ملف موجود مسبقا مع حذف جميع محتوياته ليكون حجمه زيرو بايت .. ولابد وأن يكون نوع العملية علي الملف هي GENERIC_WRITE في هذه الحالة

dwFlagsAndAttributes - 6

يحدد خصائص وعلامات الملف.. ومن الممكن الجمع بين خصائص عديدة منها:

الخاصية	المعني
FILE_ATTRIBUTE_ARCHIVE	ملف أرشيف يستخدم غالبا كملف احتياطي
FILE_ATTRIBUTE_NORMAL	ملف عادي ليس به اي خصائص ولا يجوز الجمع مع خاصية اخري إذا تم اختيار هذه الخاصية
FILE_ATTRIBUTE_READONLY	ملف للقراءة فقط لا يسمح بالكتابة أو الحذف
FILE_ATTRIBUTE_HIDDEN	ملف مخفي عند استعراض المجلد الموجود به
FILE_ATTRIBUTE_SYSTEM	ملف نظامي

وتوجد علامات Falgs للملفات قم بالنظر فيها من مرجع Win32

hTemplateFile - 7

يحدد مقبض ملف قالب يتم اخذ الخصائص Attributes منه .. ولن نستخدمه ونضع قيمته NULL.

وكما ذكرنا .. عند نجاح دالة CreateFile فإنها تعود بقيمة مقبض الملف .. بينما إذا فشلت فستعود بـINVALID_HANDLE_VALUE وقيمتها (1-).

❖ **<u>دالتي ReadFile,WriteFile</u>** :

تستخدم الدالة ReadFile للقراءة من الملف .. والدالة WriteFile للكتابة علي الملف.. وكلتا الدالتين لهما نفس المعاملات.

The **ReadFile** function reads data from a file, starting at the position indicated by the file pointer. After the read operation has been completed, the file pointer is adjusted by the number of bytes actually read, unless the file handle is created with the overlapped attribute. If the file handle is created for overlapped input and output (I/O), the application must adjust the position of the file pointer after the read operation.

```
BOOL ReadFile(
HANDLE hFile,                        // handle of file to read
LPVOID lpBuffer,                     // address of buffer that receives data
DWORD nNumberOfBytesToRead,          // number of bytes to read
LPDWORD lpNumberOfBytesRead,         // address of number of bytes read
LPOVERLAPPED lpOverlapped            // address of structure for data
);
```

The **WriteFile** function writes data to a file and is designed for both synchronous and asynchronous operation. The function starts writing data to the file at the position indicated by the file pointer. After the write operation has been completed, the file pointer is adjusted by the number of bytes actually written, except when the file is opened with FILE_FLAG_OVERLAPPED. If the file handle was created for overlapped input and output (I/O), the application must adjust the position of the file pointer after the write operation is finished.

```
BOOL WriteFile(
HANDLE hFile,                        // handle to file to write to
LPCVOID lpBuffer,                    // pointer to data to write to file
DWORD nNumberOfBytesToWrite,         // number of bytes to write
LPDWORD lpNumberOfBytesWritten,      // pointer to number of bytes written
LPOVERLAPPED lpOverlapped            // pointer to structure
);
```

1 - hFile :

مقبض الملف الذي حصلنا عليه جراء استخدام دالةCreateFile.

2 - lpBuffer :

مؤشر لمخزن البيانات الذي سيستقبل بيانات القراءة من الملف .. أو سيكون به البيانات التي ستتم كتابتها إلي الملف.

3 - nNumberOfBytesToRead /nNumberOfBytesToWrite :

قيمة عددية نصع بها عدد البايتات التي نريد قرائتها من الملف .. أو عدد البايتات التي نريد كتابتها إلي الملف.

4 - lpNumberOfBytesRead/lpNumberOfBytesWritten :

مؤشر لمتغير ستخزن به الدالة عدد البايتات المقروءة أو المكتوبة بعد تنفيذ أي من الدالتين.

5 - LPOVERLAPPED :

مؤشر لبنية OVERLAPPED .. ولن نحتاج لاستخدام هذا المعامل في برنامجنا ونضع قيمته NULL .. ولكنه معامل هام .. حيث يستخدم في عملية التزامن في الادخال والإخراج Input/Output مع دالتي ReadFile و WriteFile .

فمثلا إذا قمنا باستدعاء الدالة ReadFile بدون استخدام المعامل OverLappeed أي Not OverLapped في عملية الادخال Input .. فستدخل الدالة في مود بلوك Blocking حتي يتم قراءة كافة البايتات التي طلبتها من الملف .. ولا يتم تنفيذ أي تعليمات أخري لحين الانتهاء من تلك القراءة.

تطلق علي هذه العملية المصطلحsynchronous I/O ..

بينما إذا كانت العملية Asynchronous أو Overlapped فستتم القراءة وتنفيذ اي عمليات بدون اكتمال عملية القراءةNon Blocking.

عند نجاح أي من الدالتين فستعود بقيمة غير الصفر .. بينما إذا فشلت الدالة فستعود بالقيمة صفر.

❖ **<u>دالة GetFileSize</u> :**

تستخدم تلك الدالة في معرفة حجم الملف ..

```
The GetFileSize function retrieves the size, in bytes, of the
specified file.

 DWORD GetFileSize(
  HANDLE hFile,              // handle of file to get size of
  LPDWORD lpFileSizeHigh // address of high-order word for file size
 );
```

المعامل الأول hFile وهو مقبض الملف ..

المعامل الثاني lpFileSizeHigh وهو معامل اختياري .. يأخذ عنوان متغير بحجم DWORD في حالة إذا كان حجم الملف كبير.. حيث تعود الدالة بحجم الملف بالبايت .. وعند تجاوز القيمة المعادة الحجم [2^32] .. فيتم استخدام المعامل lpFileSizeHigh كـ High Order للقيمة المعادة في تخزين الحجم الزائد .. حيث تكون القيمة المعادة وهي بحجم DWord الـ Low Order وبهذا يكون لدينا حجم بطول [2^64] .. وتوجد دالة Extended منها باسم GetFileSizeEx يمكنك البحث عنها.

الخلاصة أنه لا حاجة لاستخدام هذا المعامل فقط قم بوضع قيمته NULL.

- **دالة SetFilePointer :**

تستخدم هذه الدالة في تحريك مؤشر الملف لموضع معين..وتفيد عندما نريد قراءة بيانات من موضع معين بالملف .. أو الكتابة في موضع معين.

```
The SetFilePointer function moves the file pointer of an open file.
DWORD SetFilePointer(
HANDLE hFile,                   // handle of file
LONG lDistanceToMove,           // number of bytes to move file pointer
PLONG lpDistanceToMoveHigh,  // address of high-order word of distance to
move
DWORD dwMoveMethod          // how to move
);
```

المعامل الأول hFile وهو مقبض الملف ..

المعامل الثاني lDistanceToMove .. وهو قيمة تحدد عدد البايتات المراد تحركها .. يستقبل قيمة موجبة إذا أردنا تحريك المؤشر للأمام .. وقيمة سالبة إذا أردنا تحريك المؤشر للخلف.
المعامل الثالث lDistanceToMoveHigh وهو قيمة مخصصة للملفات ذات الأحجام الكبيرة .. عندما نريد تحريك المؤشر في ملف بحجم64بت..ولا حاجة لاستخدام هذا المعامل ويتم وضع قيمته NULL.
المعامل الرابع dwMoveMethod وهي يحدد موضع بداية التحريك لمؤشر الملف .. وله 3 مواضع :

القيمة	المعني
FILE_BEGIN	موضع البدء هو زيرو .. أي بداية الملف
FILE_CURRENT	موضع البدء هو الموضع الحالي للمؤشر
FILE_END	موضع البدء هو نهاية الملف

وبالتالي مثلا إذا أردنا قراءة أخر 10 بايتات في ملف .. فإننا نقوم بوضع مؤشر الملف علي نهاية الملف FILE_END مع تمرير القيمة(10-) للإزاحة العكسية في المعامل الثاني ..
ومن ثم نقوم بقراءة البيانات بواسطة الدالة ReadFile مع وضع عدد البايتات التي نريد قرائتها في المعامل الثالث للدالة nNumberOfBytesToRead والذي يحدد عدد البايتات المقروءة.

❖ <u>دالة CloseHandle</u> :

بعد الانتهاء من عمليات القراءة والكتابة .. نقوم بتحرير الملف Release وإغلاق المقبض الذي حصلنا عليه من جراء دالة CreateFile.. وذلك بواسطة الدالة CloseHandle حيث نمرر لها قيمة المقبض hFile وستقوم بإغلاق الملف.

The **CloseHandle** function closes an open object handle.

```
BOOL CloseHandle(
  HANDLE hObject        // handle to object to close
);
```

توجد مجموعة دوال أخري ربما تود الإطلاع عليها من مرجع الـAPI مثل الدوال التالية :

GetFileInformationByHandle :

وهي تعود بمعلومات الملف بدلالة المقبض .. ويتم تخزين بيانات الملف في البنية BY_HANDLE_FILE_INFORMATION :

```
BY_HANDLE_FILE_INFORMATION STRUCT
  dwFileAttributes          DWORD     ?
  ftCreationTime            FILETIME  <>
  ftLastAccessTime          FILETIME  <>
  ftLastWriteTime           FILETIME  <>
  dwVolumeSerialNumber      DWORD     ?
  nFileSizeHigh             DWORD     ?
  nFileSizeLow              DWORD     ?
  nNumberOfLinks            DWORD     ?
  nFileIndexHigh            DWORD     ?
  nFileIndexLow             DWORD     ?
BY_HANDLE_FILE_INFORMATION ENDS
```

CopyFile :

وتقوم بنسخ الملف إلي ملف جديد.

MoveFile :

وتقوم بنقل الملف إلي ملف جديد.

DeleteFile :

وتقوم بحذف الملف.

CreateDirectory :

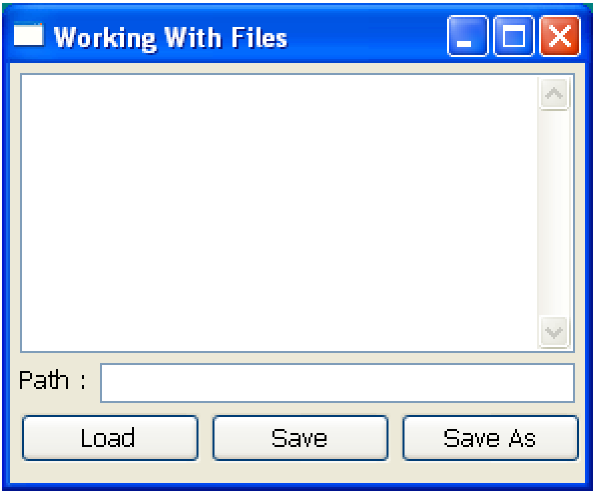
تستخدم لإنشاء مجلد جديد.

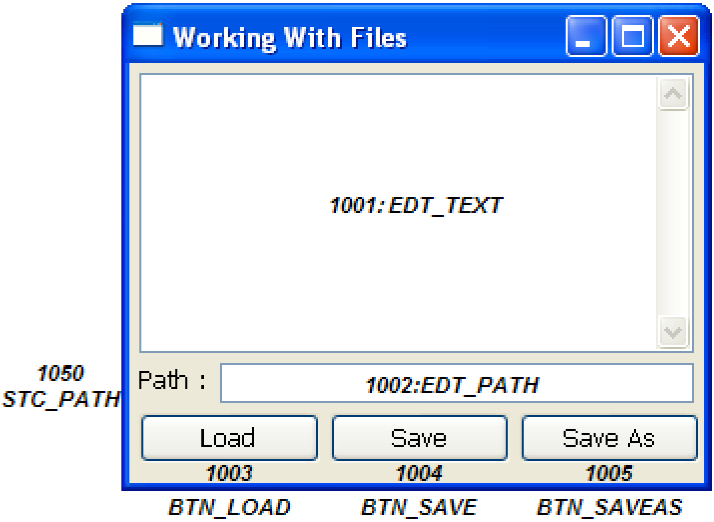
RemoveDirectory :

تستخدم لحذف مجلد.

2 – تصميم البرنامج

سنقوم بتصميم برنامج بسيط لاستعراض وحفظ الملفات النصية .. ستكون الواجهة علي الشكل التالي :





اسم المشروع هو WorkingWithFiles .. وبه صندوق نص EDT_TEXT مفعل به الخاصية MultiLine مع تمكين شريط التمرير الرأسيVertical ScrollBar وصندوق نص أخر EDT_PATH لوضع مسار الملف به .. ولن نستخدم صندوق الفتح أو الحفظ الحواري Open /Save Dialog في هذا المثال .. ونتركه لفصل قادم.

وبعد وضع مسار الملف .. يمكن الضغط علي زر Load لعرض محتويات الملف في صندوق النص EDT_TEXT .. ومن ثم يمكن التعديل علي محتويات الملف .. والحفظ بواسطة الزر Save .. أو يمكن كتابة مسار جديد للملف واختيار Save As لحفظ ملف جديد غير موجود مسبقا..

لاحظ أنه تم مراعاة تغيير حجم النافذة في هذا المشروع .. أي أن صناديق الإدخال تتغير حجمها وموضعها بتكبير أو تصغير النافذة .. كذلك يتغير الموضع الرأسي لأزرار التحكم الثلاث بتغيير حجم النافذة.. كما تم تحجيم حجم النافذة بحيث لا يقل عن ارتفاع وعرض معينين مسبقا.

وسنري كيفية تغيير حجم وموضع الأدوات أثناء الـ Sizing عندما نتطرق لكود البرنامج.

الآن نستعرض سويا محتويات الملفWorkingWithFiles.Inc

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib

DlgProc     PROTO   :HWND,:UINT,:WPARAM,:LPARAM

```

```
.const
IDD_DIALOG1         equ 101
EDT_TEXT            equ 1001
EDT_PATH            equ 1002
BTN_LOAD            equ 1003
BTN_SAVE            equ 1004
BTN_SAVEAS          equ 1005
STC_PATH            equ 1050
MaxFileName          = 255
MaxFileSize          = 65535
;####################################
.data
bPath                      db MaxFileName DUP(?)
buffer                     db MaxFileSize DUP(?)
bNull                      db NULL
MSG_PATH_MISSING           db "File Path Is Missing",0
MSG_NOFILELOADED           db "Load The File First",0
MSG_READ_FAILED            db "Read Failed",0
MSG_WRITE_SUCCESS          db "Write Success",0
MSG_WRITE_FAILED           db "Write FAILED",0
MSG_PROJECT_NAME           db "Working With Files"
dWidth                     dd ?
dHeight                    dd ?
dTop                       dd ?
dLeft                      dd ?
nBytes                     dd 0
dLength                    dd 0
rc                         RECT <>
;####################################
.data?
hInstance                  dd ?
hText                      dd ?
hBtnLoad                   dd ?
```

```
49 hBtnSave              dd ?
50 hBtnSaveAs            dd ?
51 hStcPath              dd ?
52 hPath                 dd ?
53 hFile                 dd ?
54
55 ;####################################
```

السطر 24 : يستخدم MaxFileName لتحديد طول مسار الملف.

السطر 25 : يستخدم MaxFileSize لتحديد حجم الملف الذي نقرأه أو نكتبه.

السطر 41 : يستخدم كمؤشر لدالة القراءة والكتابة لتقوم الدالة بحفظ عدد البايتات التي تمت قرائتها أو كتابتها.

السطر 43 : حجز بنية من نوع RECT والمكونة من التالي ..

The **RECT** structure defines the coordinates of the upper-left and lower-right corners of a rectangle.

```
typedef struct _RECT {    // rc
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT;
```

حيث سنخزن بها إحداثيات النافذة الأربعه .. كلا من قمة النافذة من الشاشة والقاع من الشاشة .. وبالتالي عند طرح القيمتين نحصل على ارتفاع النافذة .. كذلك يسار النافذة من الشاشة ويمين النافذة من الشاشة .. وعند طرح القيمتين نحصل على عرض النافذة.

شفرة الملف *WorkingWithFiles.Asm*

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include WorkingWithFiles.inc

.code

start:

 invoke GetModuleHandle,NULL
 mov   hInstance,eax

   invoke InitCommonControls
 invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
 invoke ExitProcess,0

;########################################################################
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

 mov   eax,uMsg
 .if eax==WM_INITDIALOG
     invoke GetDlgItem,hWin,EDT_TEXT
     mov hText,eax
     invoke GetDlgItem,hWin,EDT_PATH
     mov hPath,eax
     invoke GetDlgItem,hWin,STC_PATH
     mov hStcPath,eax
     invoke GetDlgItem,hWin,BTN_LOAD
     mov hBtnLoad,eax
     invoke GetDlgItem,hWin,BTN_SAVE
     mov hBtnSave,eax
     invoke GetDlgItem,hWin,BTN_SAVEAS
     mov hBtnSaveAs,eax

 .elseif eax==WM_COMMAND
     mov eax,wParam
     mov bx,ax
     shr eax,16
     .if ax == BN_CLICKED
```

```
    .if bx == BTN_LOAD
        invoke GetDlgItemText,hWin,EDT_PATH,addr bPath,MaxFileName
        invoke lstrcmp,addr bPath,addr bNull
        je PathMissing
        invoke CreateFile,addr bPath,GENERIC_READ,FILE_SHARE_READ,NULL,\
                              OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,NULL
        mov hFile,eax
        .if hFile==-1
           jmp LoadFailed
        .endif
        invoke ReadFile,hFile,addr buffer,MaxFileSize,addr nBytes,NULL
        .if eax==0
           jmp LoadFailed
        .endif
        invoke SetWindowText,hText,addr buffer
        ;invoke SetDlgItemText,hWin,EDT_TEXT,addr buffer
        ;invoke SendMessage,hText,WM_SETTEXT,NULL,addr buffer
        invoke CloseHandle,hFile

    .elseif bx == BTN_SAVE
        invoke lstrcmp,addr bPath,addr bNull
        je NoFileLoaded
        invoke CreateFile,addr bPath,GENERIC_WRITE,FILE_SHARE_READ,\
                       NULL,TRUNCATE_EXISTING,FILE_ATTRIBUTE_ARCHIVE,NULL
        mov hFile,eax
        .if hFile==-1
           jmp SaveFailed
        .endif

        invoke GetDlgItemText,hWin,EDT_TEXT,addr buffer,MaxFileSize
        invoke GetWindowTextLength,hText
        mov dLength,eax
        invoke WriteFile,hFile,addr buffer,dLength,addr nBytes,NULL
        .if eax==0
           jmp SaveFailed
        .endif
        invoke CloseHandle,hFile
        invoke MessageBox,hWin,addr MSG_WRITE_SUCCESS,\
                                  addr MSG_PROJECT_NAME,MB_OK

    .elseif bx == BTN_SAVEAS
        invoke GetDlgItemText,hWin,EDT_PATH,addr bPath,MaxFileName
        invoke lstrcmp,addr bPath,addr bNull
        je PathMissing
```

```
            invoke CreateFile,addr bPath,GENERIC_WRITE,FILE_SHARE_READ,\
                                NULL,CREATE_NEW,FILE_ATTRIBUTE_ARCHIVE,NULL
            mov hFile,eax
            .if hFile==-1
               jmp SaveAsFailed
            .endif
            invoke GetDlgItemText,hWin,EDT_TEXT,addr buffer,MaxFileSize
            invoke GetWindowTextLength,hText
            mov dLength,eax
            invoke WriteFile,hFile,addr buffer,dLength,addr nBytes,NULL
            .if eax==0
               jmp SaveFailed
            .endif
            invoke CloseHandle,hFile
            invoke MessageBox,hWin,addr MSG_WRITE_SUCCESS,\
                                 addr MSG_PROJECT_NAME,MB_OK
        .endif
      .endif

.elseif eax==WM_SIZE
   invoke GetClientRect,hWin,addr rc
   mov eax,rc.right
   mov ebx,rc.left
  sub eax,ebx
  sub eax,10
  mov dWidth,eax

    mov eax,rc.bottom
    mov ebx,rc.top
    sub eax,ebx
    sub eax,70
    mov dHeight,eax

    mov dTop,eax
    add dTop,10

   invoke MoveWindow,hText,5,5,dWidth,dHeight,TRUE

   invoke MoveWindow,hStcPath,5,dTop,40,20,TRUE
   sub dWidth,40
   invoke MoveWindow,hPath,45,dTop,dWidth,20,TRUE

    mov dWidth,90
    mov dHeight,25
    mov dLeft,5
    add dTop,25
```

```
      invoke MoveWindow,hBtnLoad,dLeft,dTop,dWidth,dHeight,TRUE
      mov eax,dWidth
      add dLeft,eax
      add dLeft,5
      invoke MoveWindow,hBtnSave,dLeft,dTop,dWidth,dHeight,TRUE
      mov eax,dWidth
      add dLeft,eax
      add dLeft,5
      invoke MoveWindow,hBtnSaveAs,dLeft,dTop,dWidth,dHeight,TRUE
    ret

  .elseif eax==WM_SIZING
       mov edx,lParam
    mov ebx,[edx].RECT.top
    mov eax,[edx].RECT.bottom
    sub eax,ebx
    .if eax<250
         add  ebx,250
      mov [edx].RECT.bottom,ebx
    .endif

    mov ebx,[edx].RECT.left
    mov eax,[edx].RECT.right
    sub eax,ebx
    .if eax<296
         add  ebx,296
      mov [edx].RECT.right,ebx
    .endif

 .elseif eax==WM_CLOSE
   invoke EndDialog,hWin,0
 .else
   mov   eax,FALSE
   ret
 .endif
 mov   eax,TRUE
 ret

  PathMissing:
    invoke MessageBox,hWin,addr MSG_PATH_MISSING,NULL,MB_OK
    invoke SetFocus, hPath
    ret
  NoFileLoaded:
    invoke MessageBox,hWin,addr MSG_NOFILELOADED,NULL,MB_OK
    invoke SetFocus, hPath
    ret
```

```
177   LoadFailed:
178     invoke MessageBox,hWin,addr MSG_READ_FAILED,\
179                                 addr MSG_PROJECT_NAME,MB_OK
180     invoke CloseHandle,hFile
181     ret
182   SaveFailed:
183     invoke MessageBox,hWin,addr MSG_WRITE_FAILED,\
184                                 addr MSG_PROJECT_NAME,MB_OK
185     invoke CloseHandle,hFile
186     ret
187   SaveAsFailed:
188     invoke MessageBox,hWin,addr MSG_WRITE_FAILED,\
189                                 addr MSG_PROJECT_NAME,MB_OK
190     invoke CloseHandle,hFile
191     ret
192
193 DlgProc endp
194
195 end start
```

2 – شرح شفرة البرنامج

السطور 23-34 : نقوم بإتيان مقابض جميع أدوات التحكم علي النافذة.

السطر 41 : نقطة بدء شفرة زر Load .

السطور 42-44 : نقوم بإتيان مسار الملف الذي قام المستخدم بإدخاله في صندوق النص وإذا لم يدخل المستخدم أي مسار فنقوم بالتنبيه بذلك وإظهار رسالة File Path Is Missing.

السطر 45 : نقوم باستدعاء دالة CreateFile في وضع القراءة ونمرر لها عنوان المسار .. وباقي المعاملات كما وضحنا في بداية الفصل.

السطر 47 : نقوم بنسخ قيمة المقبض التي عادت به الدالة من المسجل.

السطور 48-50 : نختبر هل نجحت الدالة في فتح الملف للقراءة أم لا عن طريقة قيمة المقبض.. فإذا كانت قيمته (1-) فهذا يعني فشل الدالة ونقوم بالتنبيه بذلك وإظهار رسالة "Read Failed" .. ومن الأفضل تحديد المشكلة

أكثر للمستخدم .. وإظهار أن البرنامج لم يستطع فتح المسار الذي ادخلته للقراءة.

السطر 51 : نقوم باستدعاء دالة ReadFile وتمرير مقبض البرنامج ومخزن البيانات وأقصي حجم للقراءة من الملف وكذلك عنوان المتغير الذي ستحفظ به الدالة عدد البايتات المقرؤة.

السطور 52-54 : نختبر هل نجحت عملية القراءة من الملف وذلك بمعرفة القيمة المعادة من الدالة ومقارنتها بالقيمة صفر.. فإذا تساوت فهذا يعني أن الدالة لم تنجح وبالتالي نقوم بإظهار رسالة 'Read Failed".

السطور 55-57 : إلي هنا تكون الدالة قد نجحت وبالتالي نقوم بإظهار محتوي الملف والذي تم حفظه في المخزن buffer في صندوق النص EDT_TEXT .. وتوجد أكثر من طريقة لذلك من بينها إرسال رسالة WM_SETTEXT إلي مقبض النافذة مع تمرير عنوان المخزن.

السطر 58 : إلي هنا تمت عملية القراءة بنجاح وبالتالي انتهي دور مقبض الملف فنقوم بلإغلاقه.

السطر 60 : بداية شفرة زرSave.

السطور 61-79 : هي نفس شفرة الزر Load مع اختلاف أننا نفتح الملف للكتابة ونأتي بنص صندوق الإدخال EDT_TEXT ونحفظه داخل الملف بدالة WriteFile.. وفي النهاية نظهر رسالة "Write Success".

السطر 81 : بداية شفرة الزرSave As .

السطور 82-100 : هي نفس شفرة الزر Save مع اختلاف أننا نفتح ملف جديد ونحصل علي مساره من صندوق الإدخال EDT_PATH .. ولا نقوم بالحفظ علي الملف الحالي .. وإذا كان الملف موجود مسبقا فتفشل الدالة.

السطر 104 : تستقبل النافذة إشعار WM_SIZE بعد أن يتم تغيير حجم النافذة .. وقيم معاملات الإشعار تكون كالتالي :

المعامل wParam :
- SIZE_MAXIMIZED=2: تم تكبير النافذة بملء الشاشة
- SIZE_MINIMIZED=1 : تم تصغير النافذة.
- SIZE_RESTORED=0 : تم تغيير حجم النافذة ولكن لا بملء الشاشة ولا تصغيرها.

المعامل lParam :

- أول 16 بت Low-Order من المعامل يحتوي علي عرض النافذة الجديد.
- أخر 16 بت High-Order من المعامل يحتوي علي ارتفاع النافذة الجديد.

وبالتالي إذا أردنا إتيان طول وعرض النافذة فسنستخدم الكود التالي :

```
mov eax,lParam   ; نسخ قيمة المعامل للمسجل
mov edx,eax      ; ننسخ قيمة المسجل للمسجل
and eax,0FFFFh   ; نقوم بالحصول علي أول 16 بايت
shr edx,16       ; نقوم بالحصول علي ثاني 16 بايت
```

راجع الفصل الأول لمعرفة كيف يعمل المعامل المنطقيAnd حيث :
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

وبالتالي عندما يتم تطبيق المعامل المنطقي AND بين قيمة حجمها 16 بت وهي[0FFFFh] والتي كل بت من الـ 16 قيمته True وبين قيمة المسجل وهي بحجم 32 بت .. فإن الـ 16 بت الممثلين في الـ High-Order لن تجد سوي قيم صفرية مطبق عليها المعامل المنطقيAND.

وفي النهاية سيحتوي المسجل eax علي عرض النافذة والمسجلedx علي ارتفاع النافذة..

هذه طريقة لإتيان ارتفاع وعرض النافذة .. بينما توجد طريقة أخري أردت استخدامها في البرنامج حتي يصلك مفهوم الإحداثيات بصورة أكبر .. نتناولها في السطر التالي.

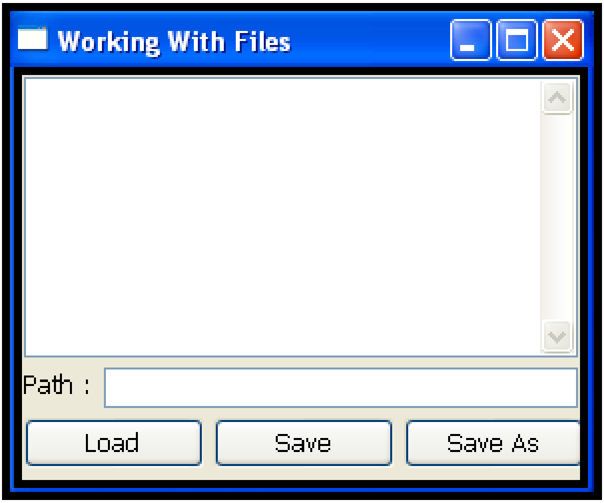
السطر 105 : تم استدعاء دالة GetClientRect والتي تقوم بإتيان إحداثيات النافذة ..

The **GetClientRect** function retrieves the coordinates of a window's client area. The client coordinates specify the upper-left and lower-right corners of the client area. Because client coordinates are relative to the upper-left corner of a window's client area, the coordinates of the upper-left corner are (0,0).

```
BOOL GetClientRect(
  HWND hWnd,          // handle of window
  LPRECT lpRect       // address of structure for client coordinates
);
```

المعامل الأول hWnd وهو مقبض النافذة المراد إتيان إحداثياتها.
المعامل الثاني lpRect وهو عنوان البنية التي ستستقبل الإحداثيات.

لاحظ أن الـ Client Rectangle Area ليس هو حجم النافذة الكلي .. وللتوضيح أكثر .. فالصورة التالية بها مستطيلان باللون الأسود .. المستطيل الداخلي هو ما يعرف بالـ Client Rectangle .. أما المستطيل الخارجي فهو حجم النافذة الكلي بالنسبة للشاشة.



السطور 106 - 110 : نقوم بمعرفة عرض النافذة وطرح القيمة 10 .. وهي تمثل 5 بكسل من يسار ويمين النافذة .. ونسخ العرض الجديد إلي المتغير dWidth.. وسيكون لدينا عرض صندوق البيانات EDT_TEXT علي النافذة.
السطور 112 – 116 : نقوم بمعرفة ارتفاع النافذة وطرح القيمة 70 وهي تمثل بـ 25 بكسل - فواصل بين كل أداة عن الأخري - وكذلك ارتفاع كلا من صندوق نص المسار وهو 20 , وايضا ارتفاع الزر وهو 25.. وبالتالي نحصل هنا علي الارتفاع المنطقي لصندوق البيانات EDT_TEXT.

السطرين 118-119 : ما زال ارتفاع صندوق البيانات لدينا في المسجل eax نقوم بنسخه - قبل تغيير قيمة المسجل eax جراء الاستدعاء الموجود في السطر 121- إلي متغير جديد dTop وإضافة 5 بكسل فاصل بين صندوق الإدخال للبيانات EDT_TEXT وللمسار EDT_PATH.. ولا ننسي الموضع الرأسي لصندوق البيانات عن قمة النافذة وهو 5 بكسل .. وبالتالي ستكون القيمة النهائية 10 بكسل .. نقوم بإضافتها إلي المتغير dTop.

السطر 121 : نستدعي دالة MoveWindow والتي تقوم بتحريك وتغيير حجم النافذة حسب الإحداثيات التي نريدها.. ونستخدمها هنا لتغيير حجم وموضع نافذة صندوق البيانات EDT_TEXT.

The **MoveWindow** function changes the position and dimensions of the specified window. For a top-level window, the position and dimensions are relative to the upper-left corner of the screen. For a child window, they are relative to the upper-left corner of the parent window's client area.

```
BOOL MoveWindow(
  HWND hWnd,          // handle of window
  int X,              // horizontal position
  int Y,              // vertical position
  int nWidth,         // width
  int nHeight,        // height
  BOOL bRepaint       // repaint flag
 );
```

ومعاملات الدالة موضحة .. فقط المعامل الأخير قيمته Boolean إما TRUE أو FALSE , وهي يحدد هل نريد إعادة رسم النافذة أم لا بعد التغيير .. نضع قيمتها TRUE وسيؤدي ذلك إلي إرسال إشعار WM_PAINT إلي النافذة.

السطور 123-125 : نقوم بضبط الموضع الرأسي لنافذة STC_PATH ثم نطرح عرض تلك النافذة وهو 40 من عرض المتغير dWidth ليصبح لدينا عرض نافذة صندوق إدخال المسار EDT_PATH.

السطور 127 – 139 : نقوم بضبط كلا من العرض dWidth والطول dHeight علي عرض وارتفاع الزر .. ونقوم بضبط الموضع الرأسي .. وكذلك الموضع الأفقي لكل زر .. حيث نزيد عرض الزر مع الفاصل بينه وبين السابق في كل مره.

السطر 142 : تستقبل النافذة إشعار WM_SIZING أثناء عمل تغيير في حجم النافذة .. وقيم معاملات هذا الإشعار تكون كالتالي :

المعامل wParam :

يحتوي هذا المعامل علي قيمه تحدد من أي حافةEdge يتم تغيير حجم النافذة منها.

WMSZ_BOTTOM	حافة القاعدة.
WMSZ_BOTTOMLEFT	حافة القاعدة اليسرى.
WMSZ_BOTTOMRIGHT	حافة القاعدة اليمني.
WMSZ_LEFT	الحافة اليسرى.
WMSZ_RIGHT	الحافة اليمنى.
WMSZ_TOP	حافة القمة.
WMSZ_TOPLEFT	حافة القمة اليسرى.
WMSZ_TOPRIGHT	حافة القمة اليمنى.

المعامل lParam :

يشير هذا المعامل إلي بنيةRECT بها إحداثيات النافذة بالنسبة للشاشة.

السطر 143 : نقوم بنسخ عنوان البنية إلي المسجلedx.

السطور 144-150 : الهدف هنا هو تحجيم ارتفاع النافذة لكي لا يقل عن 250.. فإذا تم اكتشاف أن النافذة ارتفاعها أقل من 250 بكسل.. فنجبر النافذة علي أن يكون ارتفاعها هو250.
وبالتي نطرح طول قمة النافذة من طول قاعدة النافذة ونحصل هنا علي ارتفاع النافذة .. ونقارنه بالارتفاع الذي نريده..فإذا كان أقل فنقوم بإضافة الارتفاع الذي نريده علي قمة النافذة .. ثم ننقل الناتج إلي قاع النافذة في البنية..
وطريقة الوصول لقيم معاملات البنية في الذاكرة تكون كالأتي:
[Address].Structure.Paramater

السطور 152-158 : نفس المنطق نطبقه هنا مع اختلاف قمة وقاعدة النافذة بيسار ويمين النافذة.

السطور169-191 : رسائل التنبيه التي نقوم بالقفز إليها عند فشل عملية ما.



الفصل الثامن

التعامل مع الريجستري

Access Windows Registry

1. دوال الريجستري.
2. تصميم البرنامج.
3. شرح شفرة البرنامج.

١ – دوال التعامل مع الريجستري

سنستكشف في هذا الفصل كيفية التعامل مع سجل النظام Windows Registry .. وسنتناول شرح تفصيلي للدوال التي سنستخدمها أولا .. ثم سنضع تلك الدوال موضع التنفيذ في برنامج عملي..

وكل نسخة من دوال الريجستري لها نسخة محسنة EX مخصصة لتطبيقات Win32.. وسنتعامل غالبا مع نسخ الدوال المحسنة في هذا الفصل.

❖ **دالة RegCreateKeyEx** :

تستخدم هذه الدالة في إنشاء مفتاح جديد .. وإذا كان المفتاح موجود مسبقا .. فتقوم الدالة بفتحه وإعادة مقبض هذا المفتاح.

The **RegCreateKeyEx** function creates the specified key. If the key already exists in the registry, the function opens it.

```
LONG RegCreateKeyEx(
  HKEY hKey,
  LPCTSTR lpSubKey,
  DWORD Reserved,
  LPTSTR lpClass,
  DWORD dwOptions,
  REGSAM samDesired,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  PHKEY phkResult,
  LPDWORD lpdwDisposition
);
```

للدالة 9 معاملات هم كالتالي:

8 - hKey :

وهو مقبض لمفتاح مفتوح حاليا أو أي من المقابض التالية المعرفة مسبقا:

```
HKEY_CLASSES_ROOT     =   80000000h
HKEY_CURRENT_USER     =   80000001h
```

HKEY_LOCAL_MACHINE = 80000001h
HKEY_USERS = 80000003h

والمفتاح الذي نقوم بإنشائه بواسطة RegCreateKeyEx يكون مفتاح فرعي SubKeyمن المفتاح hKey الممرر مقبضه بهذا المعامل.

9 - lpSubKey :

عنوان لنص يحدد اسم المفتاح الفرعي الذي نريد إنشائه.

10 - Reserved :

معامل محجوز .. نضع قيمته NULL.

11 - lpClass :

عنوان لنص يحدد نوع فئة المفتاح المعرفة من قبل المستخدم.. وهو معامل اختياري لن نستخدمه ونضع قيمته NULL.

12 - dwOptions :

هذا المعامل يمكن أن تكون قيمته أي من الأتي :

- REG_OPTION_NON_VOLATILE : بتخصيص تلك القيمة فإن المفتاح سيتم حفظه ولن يزول عند عملRestart للنظام.

- REG_OPTION_VOLATILE : بتخصيص تلك القيمة فإن المفتاح سيتم حفظه في الذاكرة وسيزول عند عملRestart للنظام.

13 - samDesired :

هذا المعامل يحدد الخصائص الأمنية للمفتاح .. ويمكن الجمع بين أكثر من قيمة من القيم التالية:

- **KEY_CREATE_LINK :**
ترخيص بإنشاء Symbolic Link أو Soft Link وهو عبارة عن اختصار أو مؤشر /رابط لبيانات أخري .. بحيث إذا تم حذف تلك البيانات الأخري .. فسيشير هذا المؤشر إلي لا شيء.. مثله مثل اختصار الملفات Shortcut.. ولكنه هنا يستخدم للمفاتيح .. لكي يكون مفتاح عنوان لمفتاح أخر .. وبالتأكيد لن نحتاج لاستخدامه هنا.

- **KEY_CREATE_SUB_KEY :**
ترخيص بإنشاء المفاتيح الفرعية.

- **KEY_ENUMERATE_SUB_KEYS :**
ترخيص لإتاحة عمل تعداد للمفاتيح الفرعية .. يفيد في حالة استخدام دالة RegEnumKeyEx لمعرفة جميع المفاتيح الفرعية من مفتاح ما.

- **KEY_EXECUTE :**
ترخيص لإتاحة القراءة بيانات المفتاح..ويساوي القيمةKEY_READ.

- **KEY_NOTIFY :**
ترخيص لتغيير إشعار المفتاح .. حيث نظام التشغيل به نظام حسابات بكل الأحداث Events التي تتم عليه .. والتي تستطيع استعراضها من خلال تطبيق Event Viewer المرفق مع النظام .. من ضمن تلك الأحداث تستطيع تعيين الأحداث التي تريد إشعارها للنظام عن أي تغييرات (قراءة, إنشاء , تعديل , حذف) تحدث لمفتاح معين.

- **KEY_QUERY_VALUE :**
إتاحة الاستعلام عن قيمة في المفتاح.

- **KEY_SET_VALUE :**
إتاحة وضع بيانات لقيمة.

- **KEY_WRITE :**
مكون من جمع القيم التالية :
KEY_SET_VALUE
KEY_CREATE_SUB_KEY

- **KEY_READ :**
مكون من جمع القيم التالية :
KEY_QUERY_VALUE
KEY_ENUMERATE_SUB_KEYS
KEY_NOTIFY

- **KEY_ALL_ACCESS :**
مكون من جمع القيم التالية :
KEY_QUERY_VALUE
KEY_ENUMERATE_SUB_KEYS
KEY_NOTIFY
KEY_NOTIFY

KEY_CREATE_LINK
KEY_SET_VALUE

وسنستخدم لهذا المعامل القيمة الأخيرة KEY_ALL_ACCESS لكي تتاح لنا جميع التراخيص.

14- lpSecurityAttributes :

مؤشر لبنية SECURITY_ATTRIBUTES يحدد إذا كان القيمة المعادة لمقبض المفتاح يمكن أن تورث لتطبيقات فرعية .. لن نستخدم هذا المعامل ونضع قيمته NULL.

15- phkResult :

مؤشر لمتغير والذي سيستقبل مقبض المفتاح الذي تم إنشائه أو فتحه.

16- lpdwDisposition :

مؤشر لمتغير سيستقبل إحدي القيمتين التاليتين :

- **REG_CREATED_NEW_KEY**
  المفتاح لم يكن موجود مسبقا وتم إنشائه
- **REG_OPENED_EXISTING_KEY**
  المفتاح كان موجود مسبقا وتم فتحه.

عند نجاح الدالة فستعود بالقيمة ERROR_SUCCESS وتساوي صفر .. بينما إذا فشلت .. فستعود بقيمة غير الصفر ..

❖ **دالة RegOpenKeyEx** :

تستخدم لفتح مفتاح موجود مسبقا.

The **RegOpenKeyEx** function opens the specified key.

```
LONG RegOpenKeyEx(
  HKEY hKey,              // handle of open key
  LPCTSTR lpSubKey,       // address of name of subkey to open
  DWORD ulOptions,        // reserved
  REGSAM samDesired,      // security access mask
  PHKEY phkResult         // address of handle of open key
 );
```

جميع معاملات الدالة تم تناولها في معاملات دالة RegCreateKeyEx فقط المعامل الثالث محجوز ونضع قيمته بصفر.

وعند نجاح الدالة فإنها تعود أيضا بالقيمة ERROR_SUCCESS وإلا ستعود بقيمة غير الصفر.

❖ **دالة RegDeleteKey** :

تستخدم لحذف مفتاح بجميع البيانات التي يحتويها .. ولكن لا يستطيع حذف المفتاح إذا كان محتوي علي مفاتيح فرعيةSubKeys.

```
LONG RegDeleteKey(
  HKEY hKey,              // handle of open key
  LPCTSTR lpSubKey        // address of name of subkey to delete
 );
```

المعامل hKey وهو يأخذ قيمة المقبض للمفتاح المفتوح حاليا .. أو أي قيمة من قيم المقابض المحجوزة مسبقا.

المعامل lpSubKey وهو عنوان لمتغير نصي به اسم المفتاح الفرعي المراد حذفه.

عند نجاح الدالة .. تعود بالقيمة ERROR_SUCCESS وإلا ستعود بقيمة غير الصفر.

❖ دالة RegCloseKey :

بعد انتهاء حاجتنا للمفتاح .. يجب أن نغلقه ونحرر المقبض الذي قمنا بحجزه.
ونستخدم لذلك الدالة RegCloseKey .. حيث نقوم بتمرير قيمة المقبض المفتوح لهذه الدالة.

The **RegCloseKey** function releases the handle of the specified key.

```
LONG RegCloseKey(
  HKEY hKey         // handle of key to close
 );
```

Parameters

hKey
Identifies the open key to close.

❖ دالة RegSetValueEx :

تستخدم لإنشاء وتخزين بيانات القيم للمفاتيح.

The **RegSetValueEx** function stores data in the value field of an open registry key. It can also set additional value and type information for the specified key.

```
LONG RegSetValueEx(
  HKEY hKey,                 // handle of key to set value for
  LPCTSTR lpValueName,       // address of value to set
  DWORD Reserved,            // reserved
  DWORD dwType,              // flag for value type
  CONST BYTE *lpData,        // address of value data
  DWORD cbData               // size of value data
 );
```

للدالة 6 معاملات هم كالتالي:

1 - hKey :

وهو مقبض لمفتاح مفتوح حاليا بواسطة الدالة RegCreateKeyEx أو الدالة RegOpenKeyEx أو أي من المقابض التالية المحجوزة مسبقا:

HKEY_CLASSES_ROOT	=	80000000h
HKEY_CURRENT_USER	=	80000001h
HKEY_LOCAL_MACHINE	=	80000001h
HKEY_USERS	=	80000003h

2 - lpValueName :

عنوان لمتغير نصي يحتوي علي اسم القيمة المراد إضافتها للمفتاح .. وإذا كان الاسم موجود مسبقا فيتم التغيير في القيمة نفسها بدون إضافة .. وإذا وضعت القيمة NULL لهذا المعامل فسيقوم بإلحاق بيانات القيمة بالقيمة Default بالمفتاح.

3 - Reserved :

معامل محجوز نضع قيمته صفر.

4 - dwType :

معامل يحدد نوع بيانات القيمة المراد تخزينها .. ويأخذ أي من القيم التالية:

=	القيمة	المعني
0	REG_NONE	نوع القيمة غير محدد
4	REG_DWORD	نوع رقمي 32 بت
1	REG_SZ	نوع نصي منتهي بـNULL
2	REG_EXPAND_SZ	نوع نصي منتهي بـNULL يستخدم مع النصوص المحتوية علي متغيرات البيئة Environment Variables مثل"%WinDir%"-"%Temp%"
7	REG_MULTI_SZ	نصي متعدد السطور - مصفوفة من النصوص كل نص ينتهي بـNULL *String1*\0*String2*\0*String3*\0*LastString*\0\0
3	REG_BINARY	نوع القيمة بالصيغة الثنائية

5 - lpData :

مؤشر للمتغير الموجود به البيانات المراد تخزينها لاسم القيمة المحدد.

6 - cbData :

قيمة عددية تحدد الحجم بالبايت المراد تخزينه .. فمثلا إذا كنا نخزن قيمة من النوع REG_DWORD .. فيكون الحجم 4 بايت .. وإذا كنا نخزن قيمة نصية من النوع REG_SZ أو REG_EXPAND_SZ أو REG_MULTI_SZ فيكون الحجم بطول النص المؤشر إليه في المعامل lpData بالإضافة إلي بايت حرف إنهاء النص NULL.

عند نجاح الدالة تعود بالقيمة ERROR_SUCCESS وإلا ستعود بقيمة غير الصفر.

❖ **<u>دالة RegQueryValueEx</u> :**

تستخدم لإتيان نوع وبيانات لاسم قيمة معينة ..

The **RegQueryValueEx** function retrieves the type and data for a specified value name associated with an open registry key.

```
LONG RegQueryValueEx(
  HKEY hKey,             // handle of key to query
  LPTSTR lpValueName,    // address of name of value to query
  LPDWORD lpReserved,    // reserved
  LPDWORD lpType,        // address of buffer for value type
  LPBYTE lpData,         // address of data buffer
  LPDWORD lpcbData       // address of data buffer size
 );
```

جميع المعاملات قد رأيتها من قبل في دالة RegSetValueEx الجديد هنا .. أننا نستقبل كلا من :

- نوع القيمة والممثل بالمؤشر lpType الذي يشير لعنوان متغير عددي يتم وضع نوع القيمة به.
- بيانات القيمة والممثل بالمؤشر lpData الذي يشير لعنوان متغير نصي يتم وضع البيانات المستقبلة به.
- حجم القيمة والممثل بالمؤشر lpcbData الذي يشير

وعند نجاح الدالة تعود بالقيمة ERROR_SUCCESS وإلا ستعود بقيمة غير الصفر.

❖ **دالة RegDeleteValue** :

تستخدم لحذف اسم قيمة من مفتاح محدد عن طريق مقبضه والذي نمرره في المعامل الأول .. بالإضافة إلي عنوان اسم القيمة المراد حذفها والذي نمرره في المعامل الثاني.

The **RegDeleteValue** function removes a named value from the specified registry key.

```
LONG RegDeleteValue(
  HKEY hKey,                // handle of key
  LPCTSTR lpValueName       // address of value name
 );
```

وتعود الدالة عند تنفيذها بنجاح بالقيمةERROR_SUCCESS وإلا ستعود بقيمة أخري غير الصفر.

2- تصميم البرنامج

سنقوم الآن بتصميم برنامج يغطي كل الدوال السابقة .. وواجهة البرنامج ستكون علي الشكل التالي..



اسم المشروع هو Registry وبه ثمان أزره تحكم موضحة في الشكل السابق .. بالإضافة إلي أداة Static نظهر بها نتائج التنفيذ .. بدلا من عرض النتائج في رسائل مزعجة ..

النتائج تظهر بألوان مختلفة تبعا للعملية .. فإذا تمت العملية بنجاح .. يظهر النص **باللون الأخضر** .. وإذا فشلت يظهر **باللون الأحمر** .. غير ذلك يظهر النص في الأداة STC_ STATUS باللون الأبيض مع خلفية سوداء لكل نتيجة.

ونستعرض الآن محتويات الملف Registry.Inc :

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc
include advapi32.inc
include gdi32.inc

```

```
includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib
includelib advapi32.lib
includelib gdi32.lib

DlgProc       PROTO   :HWND, :UINT, :WPARAM, :LPARAM
SetStatus     PROTO   lpText:HWND
CreateKey     PROTO   hKey:DWORD,lpSubKey:DWORD
DeleteKey     PROTO   hKey:DWORD,lpSubKey:DWORD
SaveString    PROTO   :DWORD, :DWORD, :DWORD, :DWORD, :DWORD
ReadString    PROTO   :DWORD, :DWORD, :DWORD
SaveDword     PROTO   :DWORD, :DWORD, :DWORD, :DWORD
ReadDword     PROTO   :DWORD, :DWORD, :DWORD
KillValue     PROTO   :DWORD, :DWORD, :DWORD

_hKey struct
 HKEY_CLASSES_ROOT    DWORD   ?    ;80000000h
 HKEY_CURRENT_USER    DWORD   ?    ;80000001h
 HKEY_LOCAL_MACHINE   DWORD   ?    ;80000002h
 HKEY_USERS           DWORD   ?    ;80000003h
_hKey ends

_ValueType struct
 REG_NONE             DWORD   ?   ;0
 REG_SZ               DWORD   ?   ;1
 REG_EXPAND_SZ        DWORD   ?   ;2
 REG_BINARY           DWORD   ?   ;3
 REG_DWORD            DWORD   ?   ;4
 REG_MULTI_SZ         DWORD   ?   ;7
_ValueType ends
```

```
;##########################################
.const
IDD_DIALOG1             equ 101
BTN_CREATE_KEY          equ 1001
BTN_DELETE_KEY          equ 1002
BTN_WRITE_STRING        equ 1003
BTN_READ_STRING         equ 1004
BTN_WRITE_DWORD         equ 1005
BTN_READ_DWORD          equ 1006
BTN_KILL_VALUES         equ 1007
BTN_EXIT_ME             equ 1008
STC_STATUS              equ 1009
;##########################################
.data
hKey          _hKey <80000000h,80000001h,80000002h,80000003h>
ValueType     _ValueType <0,1,2,3,4,7>
buffer              db 50 dup(?)
bKeyName            db "Dar Albaraa",0
bKeyNameSub         db "Dar Albaraa\Books",0
bValueName          db "Assembly 32Bit",0
bValueData          db "Learn coding with WIN32ASM",0
bValueName2         db "Year",0
bValueData2         dd 2009
dValueSize          dd 0
bState              db 2
MSG_EXIT_CONFIRM        db "Do You Want To Exit..(Y/N)?",0
MSG_PROJECT_NAME        db "Access Registry",0
MSG_CREATE_KEY_S        db "The key has been created",0
MSG_CREATE_KEY_F        db "The key hasn't been created",0
MSG_Delete_KEY_S        db "The key has been deleted",0
MSG_Delete_KEY_F        db "The key hasn't been deleted",0
MSG_SAVE_STRING_S       db "The string has been saved",0
MSG_SAVE_STRING_F       db "The string hasn't been saved",0
```

```
76 MSG_READ_STRING_F   db "The string hasn't been read",0
77 MSG_SAVE_DWORD_S   db "The Dword has been saved",0
78 MSG_SAVE_DWORD_F    db "The Dword hasn't been saved",0
79 MSG_READ_DWORD_F    db "The Dword hasn't been read",0
80 MSG_DELETE_VALUE_S  db "The Values have been deleted",0
81 MSG_DELETE_VALUE_F  db "The Values haven't been deleted",0
82 ;############################################
83 .data?
84 hInstance       dd ?
85 dStatus         dd ?
86 ;############################################
```

نقوم أولا بتضمين المكتبتين advapi32 و gdi32 .. الأولي مكتبة Advanced Windows API وهي المتضمنة لدوال التعامل مع الريجستري .. والثانية وهي Graphics Device Interface والمتضمنة لدوال الجرافيك والرسم .. وسنستخدمها لتغيير لون النص في أداةSTC_ STATUS.

السطر 19 : تعريف إجراء خاص بتغيير النص في الأداة STC_STATUS.. سيأخذ معامل وحيد وهو عنوان النص.. ومعظم العناوين ستكون رسائل النتائج الموجودة في السطور 70-81.

السطر 20 : تعريف إجراء خاص بإنشاء المفاتيح.. وله معاملين .. الأول يأخذ قيمة المقبض .. والثاني يأخذ عنوان النص لاسم المفتاح الجديد.

السطر 21 : تعريف إجراء خاص بحذف المفاتيح .. وله معاملين .. الأول يأخذ قيمة المقبض .. والثاني يأخذ عنوان النص لاسم ومسار المفتاح المراد حذفه.

السطر 22 : تعريف إجراء خاص بحفظ قيمة نصية .. وله 5 معاملات .. **الأول** للمقبض , **الثاني** عنوان لاسم ومسار المفتاح المراد فتحه ووضع القيمة داخله , **الثالث** عنوان لاسم القيمة , **الرابع** عنوان لبيانات القيمة , **الخامس** لحجم بيانات القيمة بالبايت.

السطر 23 : تعريف إجراء خاص بإسترجاع بيانات قيمة نصية .. وله ثلاث معاملات .. **الأول** للمقبض , **الثاني** لاسم ومسار المفتاح المراد استرجاع القيمة منه .. **الثالث** عنوان لاسم القيمة النصية المراد معرفة بياناتها.

السطر 24 : تعريف إجراء خاص بحفظ قيمة عددية 32 بت DWORD وله أربع معاملات .. هم نفس معاملات الإجراء SaveString بدون المعامل الأخير الخاص بحجم القيمة .. حيث معروف هنا أن حجم القيمة هو4 بايت.

السطر 25 : تعريف إجراء خاص باسترجاع القيمة العددية DWORD وله ثلاث معاملات .. **الأول** للمقبض , **الثاني** عنوان لاسم ومسار المفتاح, **الثالث** عنوان لاسم القيمة المراد معرفة القيمة العددية له.

السطر 26 : تعريف إجراء خاص بحذف القيم .. وله ثلاث معاملات .. **الأول** للمقبض .. **الثاني** عنوان لاسم ومسار المفتاح .. **الثالث** عنوان لاسم القيمة.

السطور 28-33 : إنشاء بنية باسم [_hKey] لحمل المقابض المعرفة مسبقا.

السطور 35 – 42 : إنشاء بنية باسم [_ValueType] لحمل أنواع القيم .. ولم نستخدم تلك البنية في مثال البرنامج .. ولكن من الممكن أن تستخدمها في كود لك مستقبلا.

السطر 57 : نقوم بحجز متغير باسم hKey من نوع البنية [_hKey] ونقوم بملء حقول بنية المتغير بقيم المقابض.

السطر 60 : حجز متغير نصي سيكون المفتاح الرئيسي.

السطر 70 : حجز متغير نصي وسيكون هو المفتاح الفرعي من المفتاح الرئيسي السابق.

السطور 62-63 : سيتم وضع قيمة نصية باسم bValueName وبيانات bValueData في المفتاح الفرعي bKeyNameSub.

السطور 64-65 : سيتم وضع قيمة عددية باسم bValueName2 وبيانات bValueData2 في المفتاح الفرعي bKeyNameSub.

السطر 66 : المتغير dValueSize سيحمل حجم بيانات القيمة النصية bValueData.

السطر 67 : المتغير bState من النوع بايت .. قمنا بحجزه لكي يحمل ثلاث قيم بثلاث حالات هم كالأتي :

- صفر [False] إذا فشل تنفيذ أمر.
- واحد [True] إذا نجح تنفيذ أمر
- اثنين إذا كانت العملية لا تتطلب أي من الحالتين السابقتين.

السطر 68 : يحمل رسالة تأكيد خروج من البرنامج ..وسنستخدمها لتنبيه المستخدم وتخييره بين غلق البرنامج أم لا.

السطور 70-81 : نرسائل لنتائج تنفيذ العمليات التي ستظهر للمستخدم عند نجاح أو فشل عملية ما.

السطر 85 : متغير غير مهيأ ليحمل مقبض نافذةSTC_STATUS.

شفرة الملف *Registry.Asm*

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include Registry.inc

.code

start:

 invoke GetModuleHandle,NULL
 mov   hInstance,eax

 invoke InitCommonControls
 invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
 invoke ExitProcess,0

;################################################################

```

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

 mov eax,uMsg
 .if eax==WM_INITDIALOG
     invoke GetDlgItem,hWin,STC_STATUS
     mov dStatus,eax

 .elseif eax==WM_COMMAND
     mov eax,wParam
     mov edx,eax
     and eax,0FFFFh
     shr edx,16
     .if edx==BN_CLICKED
       .if eax==BTN_CREATE_KEY      ;@@@@@@@@@@@@@@@@@@@
         invoke CreateKey,hKey.HKEY_CURRENT_USER,addr bKeyName
         .if bState==TRUE
            invoke SetStatus,addr MSG_CREATE_KEY_S
            invoke CreateKey,hKey.HKEY_CURRENT_USER,addr bKeyNameSub
         .else
            invoke SetStatus,addr MSG_CREATE_KEY_F
         .endif

       .elseif eax==BTN_DELETE_KEY  ;@@@@@@@@@@@@@@@@@@@
           invoke DeleteKey,hKey.HKEY_CURRENT_USER,addr bKeyNameSub
           invoke DeleteKey,hKey.HKEY_CURRENT_USER,addr bKeyName
           .if bState==TRUE
              invoke SetStatus,addr MSG_Delete_KEY_S
           .else
              invoke SetStatus,addr MSG_Delete_KEY_F
           .endif

       .elseif eax==BTN_WRITE_STRING ;@@@@@@@@@@@@@@@@@@@
          invoke lstrlen,addr bValueData
          mov dValueSize,eax
          inc dValueSize
          invoke SaveString,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                             addr bValueName,addr bValueData,dValueSize
          .if bState==TRUE
             invoke SetStatus,addr MSG_SAVE_STRING_S
          .else
             invoke SetStatus,addr MSG_SAVE_STRING_F
          .endif
```

```
      .elseif eax==BTN_READ_STRING ;@@@@@@@@@@@@@@@@@@@@
        invoke ReadString,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                              addr bValueName
        .if bState==TRUE
          mov bState,2
          invoke SetStatus,addr buffer
        .else
          invoke SetStatus,addr MSG_READ_STRING_F
        .endif

      .elseif eax==BTN_WRITE_DWORD ;@@@@@@@@@@@@@@@@@@@@
        invoke SaveDword,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                              addr bValueName2,addr bValueData2
        .if bState==TRUE
          invoke SetStatus,addr MSG_SAVE_DWORD_S
        .else
          invoke SetStatus,addr MSG_SAVE_DWORD_F
        .endif

      .elseif eax==BTN_READ_DWORD  ;@@@@@@@@@@@@@@@@@@@@
        invoke ReadDword,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                  addr bValueName2
        .if bState==TRUE
          mov bState,2
          invoke SetStatus,addr buffer
        .else
         invoke SetStatus,addr MSG_READ_DWORD_F
        .endif

      .elseif eax==BTN_KILL_VALUES ;@@@@@@@@@@@@@@@@@@@@
          invoke KillValue,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                              addr bValueName
          .if bState==TRUE
            invoke KillValue,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                              addr bValueName2
              mov bState,1
              invoke SetStatus,addr MSG_DELETE_VALUE_S
          .else
            invoke KillValue,hKey.HKEY_CURRENT_USER,addr bKeyNameSub,\
                              addr bValueName2
            .if bState==TRUE
              mov bState,1
              invoke SetStatus,addr MSG_DELETE_VALUE_S
```

```
                .else
                  mov bState,0
                  invoke SetStatus,addr MSG_DELETE_VALUE_F
                .endif
              .endif

          .elseif eax==BTN_EXIT_ME
            invoke MessageBox,hWin,addr MSG_EXIT_CONFIRM,\
                                    addr MSG_PROJECT_NAME,MB_YESNO
            .if eax==IDYES  ; == 6
              Invoke SendMessage,hWin,WM_CLOSE,0,0
            .endif
          .endif

        .endif
    .elseif eax==WM_CLOSE
      invoke EndDialog,hWin,0
    .elseif  eax==WM_CTLCOLORSTATIC
      mov eax,dStatus
      .IF eax == lParam
       invoke  SetBkMode, wParam, TRANSPARENT
       .if bState==0      ; Failed
         invoke  SetTextColor, wParam, 0000FFh      ;Red
       .elseif bState==1 ; Success
         invoke  SetTextColor, wParam, 00FF00h      ;Green
       .else                    ; Idle
        invoke   SetTextColor, wParam, 0FFFFFFh     ;White
       .endif
       invoke CreateSolidBrush,0h
       ret
      .EndIf
    .else
      mov   eax,FALSE
      ret
    .endif
    mov   eax,TRUE
    ret

DlgProc endp
;################################################
SetStatus Proc lpText:HWND
  invoke SetWindowText,dStatus,lpText
  ret
SetStatus endp
```

```
SetStatus endp
;################################################################
CreateKey Proc phKey:DWORD,lpSubKey:DWORD
 LOCAL phkResult :DWORD
 LOCAL lpdwDisposition :DWORD
  invoke RegCreateKeyEx,phKey,lpSubKey,0,0,\
        REG_OPTION_NON_VOLATILE,KEY_ALL_ACCESS,NULL,\
        ADDR phkResult,ADDR lpdwDisposition
  .if eax==ERROR_SUCCESS
    invoke RegCloseKey,phkResult
    mov bState, 1 ;TRUE
    ret
  .endif
  mov bState, 0     ;FALSE
  ret
CreateKey endp
;################################################################
DeleteKey Proc phKey:DWORD,lpSubKey:DWORD
  invoke RegDeleteKey,phKey,lpSubKey
  .if eax==ERROR_SUCCESS
    invoke RegCloseKey,phKey
    mov bState, 1 ;TRUE
    ret
  .endif
  mov bState, 0     ;FALSE
  ret
DeleteKey endp
;################################################################
SaveString Proc phKey:DWORD,lpSubKey:DWORD,lpValueName:DWORD,\
                              lpData:DWORD,cbData:DWORD
  LOCAL phkResult :DWORD
  invoke RegOpenKeyEx,phKey,lpSubKey,0,KEY_ALL_ACCESS,addr phkResult
  .if eax==ERROR_SUCCESS
   invoke RegSetValueEx,phkResult,lpValueName,0,REG_SZ,lpData,cbData
    .if eax==ERROR_SUCCESS
      invoke RegCloseKey,phkResult
      mov bState, 1 ;TRUE
      ret
    .endif
    invoke RegCloseKey,phkResult
  .endif
   mov bState, 0      ;FALSE
 ret
```

```
SaveString endp
;###############################################
ReadString Proc phKey:DWORD,lpSubKey:DWORD,lpValueName:DWORD
 LOCAL lpType:DWORD
 LOCAL phkResult :DWORD
 LOCAL lpcbData :DWORD
  invoke RegOpenKeyEx,phKey,lpSubKey,0,KEY_ALL_ACCESS,addr phkResult
  .if eax==ERROR_SUCCESS
   invoke RegQueryValueEx,phkResult,lpValueName,0,addr lpType,\
                                   addr buffer,addr lpcbData
    .if eax==ERROR_SUCCESS
      invoke RegCloseKey,phkResult
      mov bState, 1 ;TRUE
      ret
    .endif
  .endif
   mov bState, 0       ;FALSE
 ret
ReadString endp
;###############################################
SaveDword Proc phKey:DWORD,lpSubKey:DWORD,lpValueName:DWORD,\
                      lpData:DWORD
  LOCAL phkResult :DWORD
  invoke RegOpenKeyEx,phKey,lpSubKey,0,KEY_ALL_ACCESS,addr phkResult
  .if eax==ERROR_SUCCESS
   invoke RegSetValueEx,phkResult,lpValueName,0,REG_DWORD,lpData,4
    .if eax==ERROR_SUCCESS
      invoke RegCloseKey,phkResult
      mov bState, 1 ;TRUE
      ret
    .endif
  .endif
   mov bState, 0       ;FALSE
 ret
SaveDword endp
;###############################################
ReadDword Proc phKey:DWORD,lpSubKey:DWORD,lpValueName:DWORD
 LOCAL lpType:DWORD
 LOCAL phkResult :DWORD
 LOCAL lpcbData :DWORD
 LOCAL dNum :DWORD
```

```
  invoke RegOpenKeyEx,phKey,lpSubKey,0,KEY_ALL_ACCESS,addr phkResult
  .if eax==ERROR_SUCCESS
   invoke RegQueryValueEx,phkResult,lpValueName,0,addr lpType,\
                                            addr dNum,addr lpcbData
      .if eax==ERROR_SUCCESS
          invoke RegCloseKey,phkResult
          invoke dwtoa,dNum,addr buffer
          mov bState, 1 ;TRUE
          ret
      .endif
       invoke RegCloseKey,phkResult
  .endif
   mov bState, 0       ;FALSE
 ret
ReadDword endp
;###############################################################
KillValue Proc phKey:DWORD,lpSubKey:DWORD,lpValueName:DWORD
 LOCAL phkResult :DWORD
  invoke RegOpenKeyEx,phKey,lpSubKey,0,KEY_ALL_ACCESS,\
                                       addr phkResult
  .if eax==ERROR_SUCCESS
      invoke RegDeleteValue,phkResult,lpValueName
      .if eax==ERROR_SUCCESS
           invoke RegCloseKey,phkResult
           mov bState, 1 ;TRUE
           ret
      .endif
       invoke RegCloseKey,phkResult
  .endif
   mov bState, 0       ;FALSE
  ret
KillValue endp
;###############################################################
end start
```

3- شرح شفرة البرنامج

السطر 33 : بداية شفرة زر إنشاء مفتاح جديد.

السطر 34 : نستدعي إجراء CreateKey ونمرر له مقبض HKCU وكذلك عنوان اسم المفتاح المراد إنشائه.

السطر 35-40 : بعد انتهاء إجراء CreateKey فسيعود بقيمة من اثنتين مخزنة في متغير الحالة bState .. إما أن يعود بـ True وهي أي قيمة غير الصفر .. وفي تلك الحالة سنقوم باستدعاء إجراء SetStatus ليعرض رسالة نجاح إنشاء المفتاح.. أو أن يعود بقيمة False وفي تلك الحالة سنقوم بعرض رسالة عدم نجاح إنشاء المفتاح ..
وفي حالة نجاح إنشاء المفتاح .. فنقوم بإستدعاء نفس الإجراء مره أخري لكي ينشيء لنا المفتاح الفرعي.

السطور 42-49 : تتضمن شفرة زر حذف المفاتيح .. ونقوم باستدعاء إجراء DeleteKey .. مع تمرير المقبض واسم ومسار كل مفتاح .. ونبدأ أولا بحذف المفتاح الفرعي إلي أن نصل للرئيسي.. حتي تتم عملية الحذف بنجاح.

السطور 51-62 : تتضمن شفرة حفظ القيمة النصية .. ونقوم أولا بمعرفة حجم النص المراد حفظه .. بواسطة دالة lstrlen :

The **lstrlen** function returns the length in bytes (ANSI version) or characters (Unicode version) of the specified string (not including the terminating null character).

```
int lstrlen(
  LPCTSTR lpString      // address of string to count
 );
```

وتمرير عنوان المتغير النصي لها .. حتي تعود بطول النص .. ننسخه من المسجل للمتغير dValueSize ..

ونقوم باستدعاء إجراء SaveString .. وتمرير المقبض والعنوان لاسم المفتاح الفرعي .. وكذلك العنوان لاسم القيمة المراد إنشائها .. وأيضا عنوان بيانات القيمة .. وفي الأخير نمرر حجم بيانات القيمة.

السطور 63-71 : تتضمن شفرة قراءة بيانات القيمة النصية .. حيث نقوم بإستدعاء إجراء ReadString وتمرير المقبض وعنوان اسم المفتاح وعنوان اسم القيمة المراد معرفه بياناتها .. وسيعود الإجراء ببيانات القيمة في المتغير buffer ومن ثم نقوم بعرض بيانات القيمة فيSTC_STATUS .

وقد قمنا بتمرير القيمة 2 إلي bState لكي يتم عرض النص باللون الأبيض.

السطور 73-80 : تتضمن شفرة كتابة قيمة عددية وكما ذكرنا من قبل فهي لا تختلف عن شفرة كتابة قيمة نصية سوي في معامل حجم القيمة .. لإننا نعرفه مسبقا..

السطور 82-90 : تتضمن شفرة قراءة القيمة العددية وهي أيضا لا تختلف عن شفرة قراءة القيمة النصية السابقة.

السطور 92 – 110 : شفرة حذف كلا من القيمة النصية والعددية .. توجد أكثر من طريقة لضمان حذف كلا القيمتين بصورة صحيحة وإشعار المستخدم بذلك وقد أخترت أنا أسوأها .

فقط يتم استدعاء إجراء KillValue لكلا القيمتين .. والكود الزائد يمكنك حذفه .. هو فقط لضمان أنه لا توجد قيمة لم تحذف.. إذا كان هناك خطأ في حذف أي من القيمتين.

السطور 112-118 : شفرة زر إغلاق البرنامج .. حيث نقوم بإظهار الرسالة التأكيدية .. ولمعرفة رد المستخدم علي الرسالة نفحص القيمة المعادة في المسجل eax ..والقيم التالية توضح ثوابت الأزرا:

```
IDOK            equ 1
IDCANCEL        equ 2
IDABORT         equ 3
IDRETRY         equ 4
IDIGNORE        equ 5
IDYES           equ 6
IDNO            equ 7
IDCLOSE         equ 8
IDHELP          equ 9
```

وبعد التأكد من اختيار Yes نقوم بإرسال رسالة WM_CLOSE إلي النافذة.

السطر 123 : نراقب هنا إشعار جديد وهو WM_CTLCOLORSTATIC .. حيث تقوم أداة التحكم Static بإرسال هذا الإشعار للنافذة المالكه للأداة عندما يتم رسم الأداة في كل مره ..

وبيانات معاملات الإشعار تكون كالتالي :

- **wParam :** يحتوي علي الـ HDC أو **H**andle **D**evice **C**ontext وهو أحد الأمور المهمه في عالم الـ GDI .. فالـ DC يستخدم لتحديد خصائص النص والصور التي تظهر علي الشاشة **..** ونحن هنا نحصل من هذا المعامل علي مقبض الـ DC لأداة التحكم Static.. والذي يمكننا من تغيير خصائص الأداة.. كما سنري بعد قليل.

- **lParam :** يحتوي علي مقبض الأداة.

السطر 124 : نقوم بنسخ مقبض الأداة للمسجلeax.

السطر 125 : نفحص إذا كان الاشعار خاص بأداة التحكم STC_STATUS .. وذلك في حالة إذا كان هناك أكثر من أداةstatic.

السطر 126 : نقوم باستدعاء الدالة SetBkMode والتي تقوم بالتغيير في شكل خلفية الأداة.

The **SetBkMode** function sets the background mix mode of the specified device context. The background mix mode is used with text, hatched brushes, and pen styles that are not solid lines.

```
int SetBkMode(
  HDC hdc,           // handle of device context
  int iBkMode        // flag specifying background mode
);
```

حيث نمرر مقبض الـ DC للأداة في المعامل الأول .. ونمرر القيمة TRANSPARENT لكي لا يؤثر النص أو اي رسومات علي الخلفية.

```
TRANSPARENT     equ 1
OPAQUE          equ 2
```

السطور 127-133 : نقوم بمعرفة الحالة من المتغير bState ومن ثم استدعاء الدالة SetTextColor .. والتي تقوم بتحديد لون النص ..

The **SetTextColor** function sets the text color for the specified device context to the specified color.

```
COLORREF SetTextColor(
  HDC hdc,              // handle of device context
  COLORREF crColor      // text color
 );
```

حيث نمرر في المعامل أول مقبض الـ DC للأداة .. ونمرر في المعامل الثاني قيمة اللون.

السطر 134 : نقوم بإنشاء فرشاة مصمته Solid Brush من لون واحد وهي عبارة عن Bitmap أو Map of bits .. يستخدمها الويندوز لتلوين الأجزاء الداخلية للأشكال المغلقة.

ونستخدم لذلك الدالة CreateSolidBrush .. ونمرر في المعامل الوحيد لها قيمة لون الفرشاة المراد استخدامها لملء الأجزاء الداخلية .. وقد مررنا قيمة اللون الأسود .. وستعود الدالة بقيمة تحدد الفرشاة في المسجلeax

The **CreateSolidBrush** function creates a logical brush that has the specified solid color.

```
HBRUSH CreateSolidBrush(
  COLORREF crColor      // brush color value
 );
```

وحالما يتم اختيار الفرشاة المصمتة .. فعند رسم الأداة STC_STATIC سيتم تطبيق الفرشاة عليها.

السطور 146-149 : شفرة تغيير نص الأداة STC_STATIC .. وكلما يتم تغيير نص الأداة فيتم إرسال إشعار WM_CTLCOLORSTATIC لإعادة رسم الأداة.

وباقي سطور البرنامج لا يوجد بها أي صعوبة .. فقط يستلزم مراجعة الدوال التي درسناها في مقدمة الفصل.



الفصل التاسع

الصناديق الحوارية العامة

Common Dialog Boxes

1. المفهوم وكيفية الاستخدام
2. تصميم البرنامج.
3. شرح شفرة البرنامج.

1- المفهوم وكيفية الاستخدام

أثناء كتابة مثال التعامل مع الملفات قراءة وكتابة .. لم أرد استخدام الصندوق الحواري العام Open File .. حتي نجمع معظم الصناديق الحوارية في فصل خاص به.

يوجد 6 صناديق حوارية عامة Common Dialog Boxes يوفرها نظام التشغيل عن طريق مكتبة ComDlg32 .. وهم كالتالي:

صندوق الفتح Open	:	يوفر نافذة فتح الملفات.
صندوق الحفظ Save	:	يوفر نافذة حفظ الملفات.
صندوق الخط Font	:	يوفر نافذة اختيار الخط والتحكم به.
صندوق اللون Color	:	يوفر نافذة اختيار اللون.
صندوق الطباعة Print	:	يوفر نافذة الطباعة في البرنامج.
صندوق البحث والاستبدال Find and Replace	:	يوفر نافذة تتيح للمستخدم خاصية البحث وكذلك استبدال نص بنص أخر.

سنتناول أول 4 صناديق في هذا الفصل .. وكل صندوق سابق له بنية Structure تحمل بيانات هذا الصندوق العام.

البنية OPENFILENAME : تخص كلا من صندوق الفتح والحفظ.

البنية CHOOSEFONT : تخص صندوق اختيار الخطFont.

البنية CHOOSECOLOR : تخص صندوق اختيار اللون Color.

وكل بنية لها دالة خاصة بالوظيفة التي يؤديها الصندوق فمثلا :

دالة GetOpenFileName خاصة بصندوق فتح الملفات.

دالة GetSaveFileName خاصة بصندوق حفظ الملفات.

دالة ChooseFont خاصة بصندوق اختيار الخط.

دالة ChooseColor خاصة بصندوق اختيار اللون

وسنقوم في الصفحات القادمة بتفصيل كل بنية بأعضائهاMembers والدالة الخاصة بها وسنبدأ أولا ببنية صندوق الفتح والحفظOPENFILENAME.

❖ البنية OPENFILENAME :

تحتوي تلك البنية علي معلومات يتم استخدامها من قبل دالتي GetOpenFileName و GetSaveFileName لتهيئة صندوق الفتح والحفظ الحواري.. وبعد اختيار الملف للفتح أو للحفظ .. فيتم تخزين اختيار المستخدم في تلك البنية.

محتويات تلك البنية موضحة كالآتي :

The **OPENFILENAME** structure contains information that the GetOpenFileName and GetSaveFileName functions use to initialize an Open or Save As common dialog box. After the user closes the dialog box, the system returns information about the user's selection in this structure.

```
typedef struct tagOFN { // ofn
        DWORD               lStructSize;
        HWND                hwndOwner;
        HINSTANCE           hInstance;
        LPCTSTR             lpstrFilter;
        LPTSTR              lpstrCustomFilter;
        DWORD               nMaxCustFilter;
        DWORD               nFilterIndex;
        LPTSTR              lpstrFile;
        DWORD               nMaxFile;
        LPTSTR              lpstrFileTitle;
        DWORD               nMaxFileTitle;
        LPCTSTR             lpstrInitialDir;
        LPCTSTR             lpstrTitle;
        DWORD               Flags;
        WORD                nFileOffset;
        WORD                nFileExtension;
        LPCTSTR             lpstrDefExt;
        DWORD               lCustData;
        LPOFNHOOKPROC       lpfnHook;
        LPCTSTR             lpTemplateName;
} OPENFILENAME;
```

حيث تحتوي علي 20 عضو .. نتناول أهم تلك الأعضاء الأتي :

1 - lStructSize :

عضو إجباري .. يحدد حجم البنية بالبايت ونحصل عليه عن طريق التعليمة sizeof.

2 - hwndOwner :

عضو إختياري .. يحدد مقبض النافذة التي تمتلك الصندوق .. وستكون هنا نافذة البرنامج.

3 - lpstrFilter :

عضو إختياري ..وهو مؤشر لنص يحتوي نص فلترة الملفات .. فهذا المعامل يتحكم بأنواع الملفات التي نريد عرضها تبعا لامتداد تلك الملفات .. ويوجد نموذج لعمل ذلك .. حيث النص يكون مكون من جزئين يفصل بينهم حرف NULL .. الجزء الأول يصف الامتداد .. بينما الجزء الثاني نضع به امتداد الملفات المراد عرضها فقط ..

فمثلا نستطيع عرض ملفات المنتهية بالامتداد "*.txt" وسيكون نص الفلترة بهذا الشكل :

"Text Files (*.txt)" 0 "*.txt"

الجزء الأول به وصف الملف .. ثمNULL ثم امتداد الملفات المراد عرضها.

ولعرض أكثر من امتداد في نفس الوقت .. فنستخدم semi colon (;) للفصل بين الامتدادات بهذا الشكل :

"Text Files (*.txt)" 0 "*.txt" ; "Assembly Files (*.asm)" 0 "*.asm"

وبنهاية نص الفلترة لابد وأن نضع حرفين NULL .. ويتم عرض تلك الفلترة في نافذة ComboBox أسفل الصندوق الحواري .. وإذا لم نحدد نص للفلترة فلن تظهر أي اختيارات للفلترة في الصندوق الحواري وسيتم عرض جميع الملفات.

4 - lpstrFile :

عضو إجباري .. وهو مؤشر لمخزن نصي .. يستخدم في مرحلتين .. المرحلة الأولي عند ظهور الصندوق الحواري للمستخدم فيحتوي هذا العضو علي عنوان اسم\مسار الملف المراد فتحه أو حفظه ويمكن هنا أن يكون المخزن NULL إذا لم نرد عرض اسم ملف للمستخدم .. بينما بعد اختيار المستخدم

لاسم الملف الذي يريده سواء لعملية الفتح او للحفظ فسيحتوي علي مسار واسم الملف الذي اختاره المستخدم.

5 - nMaxFile :

عضو إجباري .. وهو يحدد طول المخزن المؤشر له بواسطة العضو lpstrFile ليحدد أكبر طول لمسار الملف .. فإذا كان مسار الملف أكبر من الطول المحدد بهذا العضو تعود دالة GetOpenFileName و GetSaveFileName بالقيمة False.. وبصفه عامة فإن طول المسار يجب أن يكون في حدود 256 حرف.. وبالتالي يجب أن يكون المخزن المؤشر عليه بالعضوlpstrFile بهذا الحجم.

6 - lpstrFileTitle :

عضو إختياري .. وهو مؤشر لمخزن نصي يستقبل اسم الملف والامتداد بدون المسار.. ويمكن أن نتجاهل هذا العضو.

7 - nMaxFileTitle :

عضو إختياري..وهو يحدد طول المخزن المؤشر له بواسطة العضو lpstrFileTitle .. ويتم تجاهل هذا العضو إذا كانت قيمة العضو lpstrFileTitle هي NULL.

8 - lpstrInitialDir :

عضو إختياري .. وهو مؤشر لنص يحدد مسار المجلد الاستهلالي للملف.

9 - lpstrTitle :

عضو إختياري .. وهو مؤشر لنص شريط العنوان للصندوق الحواري العام.. حيث نص شريط العنوان الافتراضي لكلا الصندوقين هما Open, Save As .. يمكنك تغيير هذا العنوان بواسطة هذا العضو.

10 -Flag :

توجد رايات عديدة لتهيئة الصندوق الحواري العام .. ومن بعض تلك الرايات التالي:

تمكين التأشير واختيار أكثر من ملف.	OFN_ALLOWMULTISELECT
إذا كان الملف غير موجود فالراية تجعل صندوق الحوار تظهر رسالة إنشاء الملف.	OFN_CREATEPROMPT
تجعل الراية الصندوق الحواري يأخذ شكل المستعرض الجديد.	OFN_EXPLORER

تجعل الراية المستخدم يكتب اسم ملف موجود فقط وإذا لم يكن موجود فسيظهر تحذير بذلك.	OFN_FILEMUSTEXIST
يوجد صندوق اختيار أسف صندوق الحوار باسم "Open as read only" تجعل الراية هذا الصندوق يختفي.	OFN_HIDEREADONLY
إخفاء زر الشبكة.	OFN_NONETWORKBUTT
تحديد أن الملف ليس للقراءة فقط أو المجلد ليس ممنوع الكتابة به.	OFN_NOREADONLYRETURN
جعل الصندوق الحواري العام يظهر رسالة الكتابة الملف إذا كان موجود مسبقا.	OFN_OVERWRITEPROMPT
تحديد أن المستخدم يستطيع كتابة اسم ومسار صحيح فقط..وإظهار رسالة تحذير.	OFN_PATHMUSTEXIST
يجعل صندوق الاختيار Read Only مؤشر عليه .	OFN_READONLY

الآن بعد ملء الأعضاء الهامة ببنية OPENFILENAME .. نقوم بإستدعاء الدالة : GetOpenFileName

The **GetOpenFileName** function creates an Open common dialog box that lets the user specify the drive, directory, and the name of a file or set of files to open.

```
BOOL GetOpenFileName(
  LPOPENFILENAME lpofn // address of structure with initialization data
);
```

حيث تقوم بإنشاء صندوق حوار الفتح..والمعامل الوحيد لها lpOpenFileName هو مؤشر لبنية OPENFILENAME التي قمنا بملء أعضائها بالبيانات اللازمة.

وبعد أن يختار المنستخدم الملف المراد فتحه .. ويتم إغلاق صندوق الفتح الحواري .. يتم تخزين اختيار المستخدم للملف في العضو lpstrFile بالبنية والذي هو مؤشر لمخزن نصي به مسار واسم وامتداد الملف.

وعند نجاح الدالة تعود بقيمة غير الصفر..بينما إذا فشلت فتعود بالقيمة صفر.

❖ البنية CHOOSEFONT :

وهي تحتوي على معلومات يتم استخدامها من قبل دالة ChooseFont لتهيئة صندوق الخط الحواري.

The **CHOOSEFONT** structure contains information that the **ChooseFont** function uses to initialize the Font common dialog box. After the user closes the dialog box, the system returns information about the user's selection in this structure.

```
typedef struct {    // cf
    DWORD        lStructSize;
    HWND         hwndOwner;
    HDC          hDC;
    LPLOGFONT    lpLogFont;
    INT          iPointSize;
    DWORD        Flags;
    DWORD        rgbColors;
    LPARAM       lCustData;
    LPCFHOOKPROC lpfnHook;
    LPCTSTR      lpTemplateName;
    HINSTANCE    hInstance;
    LPTSTR       lpszStyle;
    WORD         nFontType;
    WORD         ___MISSING_ALIGNMENT__;
    INT          nSizeMin;
    INT          nSizeMax;
} CHOOSEFONT;
```

حيث تحتوي البنية على 16 عضو .. ونتناول أهم تلك الأعضاء كالتالي :

1 - lStructSize :

عضو إجباري .. يحدد حجم البنية بالبايت .. ونحصل عليه بتعليمةsizeof.

2 - hwndOwner :

عضو إختياري .. يحدد مقبض النافذة المالكة لصندوق الخط الحواري.

3 - lpLogFont :

عضو إجباري .. مؤشر إلي بنية من نوع LOGFONT .. وهذه البنية تحدد خصائص الخط .. مثل الطول والعرض والحجم واسم الخط والعائلة .. وسيلي الحديث عن تلك البنية بعد قليل.

4 - Flags :

رايات تتستخدم لتهيئة صندوق الخط الحواري .. وبعض تلك الرايات كالتالي:

يجعل الصندوق الحواري يظهر التأثيرات علي الخطوط .. مثل الخط المنتصف والسفلي.	CF_EFFECTS
تحديد أنه يجب اختيار خط موجود .. وإلا سيظهر رسالة تحذير.	CF_FORCEFONTEXIST
يجعل صندوق الخط يستخدم بنية LOGFONT المؤشر عليها بواسطة lpLogFont لتهيئة أدوات التحكم في صندوق الخط الحواري ببيانات أولية.	CF_INITTOLOGFONTSTRUCT
يجعل صندوق الخط الحواري يختار أحجام خطوط محددة المدي بين nSizeMin و nSizeMax.	CF_LIMITSIZE
يجعل الصندوق الحواري يسرد قائمة الخطوط المدعومة بالنظام.	CF_SCREENFONTS

5 - nSizeMin,nSizeMax :

عضو إختياري .. حيث يحددان أكبر وأقل حجم للخطوط التي يستطيع المستخدم اختيارها.

❖ البنية LOGFONT :

سبق أن ذكرنا أن تلك البنية تحدد خصائص الخط Font Attributes ..أعضاء تلك البنية موضحين كالأتي:

The **LOGFONT** structure defines the attributes of a font.

```
typedef struct tagLOGFONT { // lf
   LONG lfHeight;
   LONG lfWidth;
   LONG lfEscapement;
   LONG lfOrientation;
   LONG lfWeight;
   BYTE lfItalic;
   BYTE lfUnderline;
   BYTE lfStrikeOut;
   BYTE lfCharSet;
   BYTE lfOutPrecision;
   BYTE lfClipPrecision;
   BYTE lfQuality;
   BYTE lfPitchAndFamily;
   TCHAR lfFaceName[LF_FACESIZE];
} LOGFONT;
```

حيث :

1 - lfHeight : ارتفاع الخط.
2 - lfWidth : عرض الخط.
3 - lfWeight :يحدد ثقل الخط
4 - lfItalic : خط مائل إذا كانت القيمة True.
5 - lfUnderline : خط أفقي تحت النص إذا كانت القيمةTrue.
6 - lfStikeOut : خط يقطع النص.
7 - lfCharSet : يحدد مجموعة الحروف للغات .. للغة العربية تكون القيمة178.
8 - lfPitchAndFamily : يحد الانحدار وعائلة الخط.
9 - lfFaceName : نص لا يتجاوز32 حرف يحدد اسم الخط المستخدم.

والآن بعد ملء الأعضاء الهامة لبنية CHOOSEFONT .. نقوم بإستدعاء الدالة ChooseFont التالية :

The **ChooseFont** function creates a Font common dialog box that enables the user to choose attributes for a logical font. These attributes include a typeface name, style (bold, italic, or regular), point size, effects (underline, strikeout, and text color), and a script (or character set).

```
BOOL ChooseFont(
  LPCHOOSEFONT lpcf // pointer to structure with initialization data
);
```

حيث تقوم بإنشاء صندوق الخط الحواري ..ونمرر عنوان البنيةCHOOSEFONT لها .. وبعد إظهار الصندوق واختيار المستخدم للخط وخصائصه التي يرديها .. تعود الدالة ببيانات الخط المختار في البنيةlogfont.

وتعود الدالة عند نجاحها بقيمة غير الصفر والعكس عند فشل عمل الدالة.

❖ البنية CHOOSECOLOR :

تحتوي هذه البنية علي معلومات يتم استخدامها من قبل دالةChooseColor لتهيئة صندوق اللون الحواري.

The **CHOOSECOLOR** structure contains information the **ChooseColor** function uses to initialize the Color common dialog box. After the user closes the dialog box, the system returns information about the user's selection in this structure.

```
typedef struct {   // cc
    DWORD        lStructSize;
    HWND         hwndOwner;
    HWND         hInstance;
    COLORREF     rgbResult;
    COLORREF*    lpCustColors;
    DWORD        Flags;
    LPARAM       lCustData;
    LPCCHOOKPROC lpfnHook;
    LPCTSTR      lpTemplateName;
} CHOOSECOLOR;
```

1 - lStructSize :

عضو إجباري .. يحدد حجم البنية بالبايت .. ونحصل عليه بتعليمةsizeof.

2 - hwndOwner :

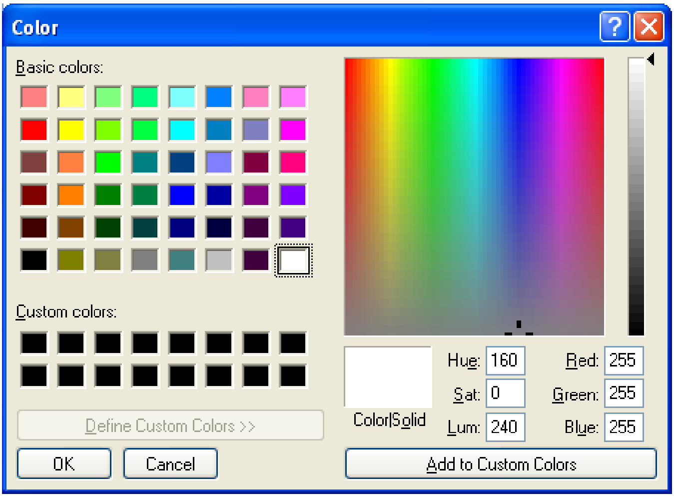
عضو إختياري .. يحدد مقبض النافذة المالكة لصندوق الخط الحواري.

3 - rgbResult :

عضو إجباري .. يستخدم هذا العضو في مرحلتين الأولي وهي استخدامه مع الراية CC_RGBINIT ليكون اللون الافتراضي المؤشر عليه عند ظهور نافذة صندوق اللون.. الثانية وهي الحصول علي نتيجة اختيار العضو للون الذي يريده.

4 - rgbResult :

عضو إجباري .. وهو مؤشر لمصفوفة من 16 حد .. كل حد قيمته DWORD 32 بت .. وكل حد يمثل لون من الـ 16 لون المخصصين Custom Colors في نافذة صندوق اللون.. حيث عندما يقوم المستخدم بتعديل الالوان المخصصة فسيقوم النظام بالتعديل في المصفوفة لحفظ الألوان المخصصة التي اختارها المستخدم



يتم تمثيل الالوان غالبا بقيم هكس في البرامج علي الصيغة RGB .. فمثلا القيمة التالية :

```
00   FF   FF   FF
00   255  255  255
00   RR   GG   BB
```

تمثل اللون الأبيض .. وإذا أردت تمثيلها عشريا فستحول القيمة الهكس 0FFFFFF إلي قيمة عشرية لتصبح 16777215.

5 - Flags :

مجموعه من الرايات التي تستخدم لتهيئة صندوق اللون الحواري

يجعل الصندوق يفتح نافذة اللون بأكملها بدون إخفاء للألوان المخصصة.	CC_FULLOPEN
يجعل النافذة مقتصرة علي الألوان الأساسية ولا يتم تنشيط زر 	CC_PREVENTFULLOPEN
يجعل صندوق الحوار يستخدم اللون المحدد في rgbResult كاختيار منشط عند فتح النافذة	CC_RGBINIT
يجعل الصندوق يظهر جميع مجموعات الألوان المتاحة في فئة الألوان الأساسية	CC_ANYCOLOR

والآن بعد ملء الأعضاء الهامة لبنية CHOOSECOLOR .. نقوم بإستدعاء الدالة ChooseColor التالية :

```
The ChooseColor function creates a Color common dialog box
that enables the user to select a color.

 BOOL ChooseColor(
  LPCHOOSECOLOR lpcc // pointer to structure with initialization data
 );
```

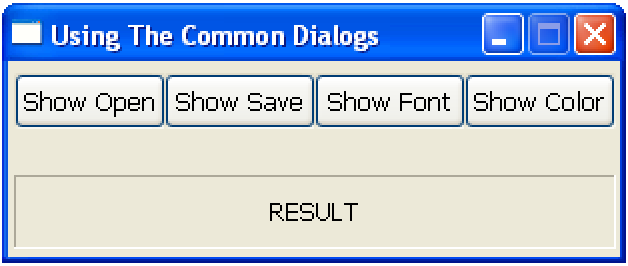
والتي تقوم بإنشاء صندوق اللون الحواري .. ونمرر في المعامل الوحيد لها مؤشر البنية CHOOSECOLOR .. وتعود بقيمة اللون الذي قام المستخدم بإختياره في العضو rgbResult.

تعود الدالة بالقيمة صفر إذا فشلت .. وتعود بقيمة غير الصفر عند نجاحها.

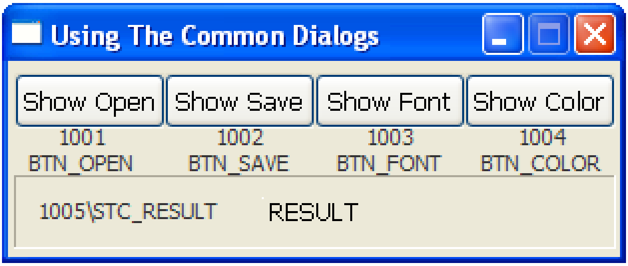
2 – تصميم البرنامج

سنقوم حاليا بتصميم برنامج يحتوي علي الصناديق الحوارية العامة التي درسناها في هذا الفصل.

واجهه البرنامج ستكون علي الشكل التالي :



وموضح عليها أرقام المعرفات للأدوات والأسماء كالتالي:



اسم المشروع هو CommonDialog .. ويحتوي علي صندوق حوراري باسم CommonDialog.dlg وملف الشفرة CommonDialog.Asm وملف التضمينات CommonDialog.Inc .. ويحتوي الصندوق الحواري علي أربعة أزرار .. وأداة static لعرض نتيجة تنفيذ الأزرار بها.
زر Open يستخدم لفتح صندوق الحوار العام Open وبعد اختيار الملف المراد فتحه سيظهر اسم ومسار الملف في STC_RESULT.

زر Save يستخدم لفتح صندوق الحوار العام Save وبعد اختيار اسم ومسار الملف المراد حفظه سيظهر في STC_RESULT.
زر Font يستخدم لفتح صندوق الحوار العام Font وبعد اختيار خصائص الخط سيتم تطبيق خصائص الخط علي STC_RESULT.

زر Color يستخدم لفتح صندوق الحوار العامColor وبعد اختيار اللون .. سيتم تطبيق قيمة اللون عليSTC_RESULT..

والآن نستعرض محتويات الملفCommonDialog.Inc

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include comdlg32.inc
include gdi32.inc
include masm32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib comdlg32.lib
includelib gdi32.lib
includelib masm32.lib

DlgProc     PROTO  :HWND,:UINT,:WPARAM,:LPARAM

.const
IDD_DIALOG1     equ 101
BTN_OPEN        equ 1001
BTN_SAVE        equ 1002
BTN_FONT        equ 1003
BTN_COLOR       equ 1004
STC_RESULT      equ 1005
;###########################################
```

```
28 .data
29 FilterString       db "All Files",0,"*.*",0,0
30 bgColor            dd  0D8E9ECh ; Gray Window Color
31 fgColor            dd   0h ; Default Black Text
32 ;###########################################
33 .data?
34 hInstance          dd ?
35 dResult            dd ?
36 ofn                OPENFILENAME <>
37 cc                 CHOOSECOLOR <>
38 cf                 CHOOSEFONT <>
39 logfont            LOGFONT <>
40 FileName           db 255 dup(?)
41 CustomColors       dd 16 dup(?)
42 ;###########################################
```

في البداية تم تضمين مكتبة comdlg32 .. وتم حجز معرفات الأدوات.

السطر 29 : تهيئة نص فلتر عرض الملفات .. وسنقوم بعرض جميع الملفات "*.*" سواء لصندوق الفتح أو الحفظ.

السطر 30 : متغير عددي سيحمل قيمة اللون الرمادي للخلفية الافتراضي.

السطر 31 : متغير عددي سيحمل قيمة اللون الذي سيختاره المستخدم .. وقمنا بتهيئته في البداية ليحمل لون النص الأسود الافتراضي.

السطر 35 : متغير عددي سيحمل قيمة مقبض STC_RESULT.

السطر 36 : متغير لبنية OPENFILENAME .. المستخدمه في صندوق الفتح والحفظ.

السطر 37 : متغير لبنية CHOOSECOLOR المستخدمة في صندوق اللون.

السطر 38 : متغير لبنية CHOOSEFONT المستخدمة في صندوق الخط.
السطر 39 : متغير لبنية LOFFONT والتي ستحمل خصائص الخط المختار.

السطر 40 : متغير نصي بطول 255 بايت ليحمل مسار الملف.

السطر 41 : مصفوفة مكونة من 16 متغير عددي DWORD لتحمل قيم الألوان المخصصة في صندوق اللون.

شفرة الملف *Registry.Asm*

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none   ;case sensitive

include CommonDialog.inc

.code

start:

  invoke GetModuleHandle,NULL
  mov    hInstance,eax

    invoke InitCommonControls
  invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
  invoke ExitProcess,0

;########################################################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

  mov    eax,uMsg
```

```
.if eax==WM_INITDIALOG
      invoke GetDlgItem,hWin,STC_RESULT
      mov dResult,eax

.elseif eax==WM_COMMAND
          mov eax,wParam
          mov edx,eax
          and eax,0FFFFh
          shr edx,16
          .if edx==BN_CLICKED
            .if eax==BTN_OPEN
               mov ofn.lStructSize,sizeof ofn
                push hWin
                pop ofn.hwndOwner
                mov ofn.lpstrFilter,offset FilterString
               mov ofn.lpstrFile,offset FileName
               mov ofn.nMaxFile, 00FFFFh
               mov ofn.Flags,OFN_FILEMUSTEXIST or \
                                   OFN_PATHMUSTEXIST or \
                                   OFN_EXPLORER or OFN_HIDEREADONLY

              invoke GetOpenFileName,ADDR ofn
              invoke SetDlgItemText,hWin,STC_RESULT,addr FileName

        .elseif eax==BTN_SAVE
             mov ofn.lStructSize,sizeof ofn
              push hWin
              pop ofn.hwndOwner
              mov ofn.lpstrFilter,offset FilterString
              mov ofn.lpstrFile,offset FileName
              mov ofn.nMaxFile, 00FFFFh

```

```
            mov ofn.Flags,OFN_EXPLORER or OFN_LONGNAMES or \
                                    OFN_HIDEREADONLY

            invoke GetSaveFileName,ADDR ofn
            invoke SetDlgItemText,hWin,STC_RESULT,addr FileName


        .elseif eax==BTN_FONT
            mov cf.lStructSize,sizeof cf
             push hWin
             pop cf.hwndOwner
            mov cf.lpLogFont,offset logfont
            mov cf.Flags,CF_SCREENFONTS or CF_EFFECTS
            mov cf.nFontType,SCREEN_FONTTYPE
            invoke ChooseFont, addr cf
           invoke CreateFontIndirect,addr logfont
           invoke SendMessage, dResult, WM_SETFONT, eax, 1

           .elseif eax==BTN_COLOR
               mov cc.lStructSize,sizeof cc
               push hWin
               pop cc.hwndOwner
               push fgColor
               pop cc.rgbResult
               mov cc.lpCustColors,offset CustomColors
               mov cc.Flags,CC_ANYCOLOR or CC_RGBINIT
               invoke ChooseColor,addr cc
               .if eax!=0
                   push cc.rgbResult
                   pop fgColor
                   invoke InvalidateRect,dResult,0,TRUE
              .endif
```

```
            .endif
        .endif

.ElseIf eax==WM_CTLCOLORSTATIC
    mov eax,dResult
    .IF eax == lParam
        invoke SetBkMode,wParam,TRANSPARENT
        invoke SetTextColor,wParam,fgColor
        invoke CreateSolidBrush,bgColor
        ret
    .EndIf

.elseif eax==WM_CLOSE
      invoke EndDialog,hWin,0
.else
      mov    eax,FALSE
      ret
.endif
  mov   eax,TRUE
ret

DlgProc endp

end start
```

3 – شرح شفرة البرنامج

السطر 33 : بداية شفرة زر تشغيل صندوق الحوارOpen.

السطر 34 : نقوم بنسخ حجم البنيةofn إلي العضوlStructSize.

السطر 35 : نقوم بدفع قيمة مقبض النافذة إلي الاستاك.

السطر 36 : نقوم بإخراج قيمة المقبض ووضعه في العضوhwndOwner.

السطر 37 : نقوم بتمرير عنوان نص الفلترة للعضوlpstrFilter.

السطر 38 : نقوم بتمرير عنوان المتغيرFileName -والذي سيستقبل مسار واسم الملف - إلي العضوlpstrFile.

السطر 39 : تحديد الملفات المراد فتحها بحجم65536 بايت.

السطر 40 : إعطاء بعض الرايات المهيئة لصندوق الفتح.

السطر 44 : استدعاء الدالة GetOpenFileName وتمرير عنوان البنية لها.. ومن المنطقي هنا أن نضع كود اختبار نجاح الدالة من عدمه.. بمقارنة القيمة المعادة مع الصفر.

السطر 45 : نقوم بإرسال مسار الملف إليSTC_RESULT.

السطور 47-59 : شفرة زر إظهار صندوق الحفظ الحواري .. وهي لا تختلف كثيرا عن شفرة زر صندوق الفتح.. سوي في استدعاء دالة GetSaveFileName والتي تعود بمسار واسم الملف المراد حفظه في المتغيرFileName والذي نقوم بإرساله إلي STC_RESULT.

السطور 61-70 : شفرة زر إظهار صندوق الخط الحواري .. نقوم فيها بعد ملء أعضاء البنية بالبيانات اللازمة .. بإستدعاء الدالة ChooseFont مع تمرير عنوان البنية لها.. وبعد أنا نحصل علي خصائص الخط في بنية logfont .. يوجد أمامنا طريقتين .. إما إستدعاء دالة CreateFont و نمرر لها خصائص الخط يدويا .. أو استدعاء دالة CreateFontIndirect ونمرر لها مؤشر لبنية logfont وتقوم بإنشاء الخط لنا.

The **CreateFont** function creates a logical font that has specific characteristics. The logical font can subsequently be selected as the font for any device.

```
HFONT CreateFont(
  int nHeight,                       // logical height of font
  int nWidth,                        // logical average character width
  int nEscapement,                   // angle of escapement
  int nOrientation,                  // base-line orientation angle
  int fnWeight,                      // font weight
  DWORD fdwItalic,                   // italic attribute flag
  DWORD fdwUnderline,                // underline attribute flag
  DWORD fdwStrikeOut,                // strikeout attribute flag
  DWORD fdwCharSet,                  // character set identifier
  DWORD fdwOutputPrecision,          // output precision
  DWORD fdwClipPrecision,            // clipping precision
  DWORD fdwQuality,                  // output quality
  DWORD fdwPitchAndFamily,           // pitch and family
  LPCTSTR lpszFace                   // pointer to typeface name string
);
```

The **CreateFontIndirect** function creates a logical font that has the characteristics specified in the specified structure. The font can subsequently be selected as the current font for any device context.

```
HFONT CreateFontIndirect(
  CONST LOGFONT *lplf        // pointer to logical font structure
);
```

المهم بعد استخدام أي من الدالتين فإنهما يعودان بقيمة مقبض الخط في المسجل eax.. ونقوم بإرسال هذا المقبض مع رسالة WM_SETFONT إلي نافذة STC_RESULT.

حيث wParam سيحتوي علي قيمة المقبض للخط .. وإذا كان هذا المعامل NULL فسيتم استخدام خط النظام الافتراضي لرسم النص ..

بينما lParam يحدد هل نريد للأداة أن تعيد رسم محتوياتها بعد إرسال رسالة تحديد الخط مباشرة .. ونضع لهذا المعامل القيمةTrue.

السطور 72-85 : شفرة زر إظهار صندوق اللون الحواري .. وبعد ملء أعضاء البنية بالبيانات اللازمة .. نقوم بإستدعاء الدالة ChooseColor لعرض صندوق اللون الحواري للمستخدم .. وبعد أن يختار المستخدم اللون الذي يريده فستعود الدالة بقيمة غير الصفر .. ونقوم بالحصول علي قيمة اللون من العضو rgbResult ونضعه في المتغير fgColor ثم نقوم بإستدعاء الدالة InvalidateRect والتي تقوم بعمل Update لمنقطة مستطيلة في النافذة لتعاد رسمها من جديد.

The **InvalidateRect** function adds a rectangle to the specified window's update region. The update region represents the portion of the window's client area that must be redrawn.

```
BOOL InvalidateRect(
  HWND hWnd,              // handle of window with changed update region
  CONST RECT *lpRect,     // address of rectangle coordinates
  BOOL bErase             // erase-background flag
);
```

المعامل الأول وهو مقبض النافذة .. **المعامل الثاني** وهو مؤشر لبنية من نوع RECT لتحمل إحداثيات المنطقة المراد إعادة رسمها في النافذة..
المعامل الثالث وهو راية تحدد هل تريد مسح خلفية تلك المنطقة أم لا.

ونمرر قيمة مقبض النافذة STC_RESULT والقيمة صفر لمؤشر البنية لعمل إعادة رسم للنافذة كلها .. والقيمة TRUE لمسح الخلفية.

السطور 89 – 96 : عند إستدعاء الدالة InvalidateRect فستقوم بإرسال إشعار WM_CTLCOLORSTATIC لنافذة الصندوق الحواري .. نستطيع هنا أن نعرف إلي أين يصل هذا الإشعار بمقارنة قيمة lParam المحتوية علي مقبض النافذة مع قيمة مقبض نافذة STC_RESULT ..
ومن ثم نقوم بتغيير لون نص الأداة .. ووضع الخلفية الرمادية .. كما درسنا في الفصل السابق.



الفصل العاشر

التصنيف الفرعي

Hook and Subclassing

1. المفهوم وكيفية الاستخدام
2. تصميم وتحليل البرنامج.

١ – المفهوم وكيفية الاستخدام

تعمل جميع تطبيقات GUI علي استقبال رسائل من نظام التشغيل وترجمة تلك الرسائل إلي أحداث نستطيع التعامل معها.

فمثلا عندما تتحرك الفأرة علي نافذة .. فيقوم نظام التشغيل بإرسال إشعار إلي تلك النافذة تبين أن الفأرة قد تحركت..وها هي إحداثيات الفأرة الجديدة.

هذه الإشعارات تستقبلها النافذة من خلال صندوق إجراء النافذة WndProc كما استعرضنا ذلك في الفصل الرابع .. أو صندوق إجراء نافذة الصندوق الحوراي DlgProc.

ولكن ماذا عن رسائل أدوات التحكم علي النافذة .. ماذا إذا أردنا مثلا تحديد هل الفأرة تتحرك علي أداة معينة .. ماذا إذا أردنا فلترة مدخلات صندوق نص ليقبل حروف معينة ويتجاهل الأخري .. ماذا إذا أردنا تمكين المستخدم من تحريك Drag أو تغيير حجم أدوات التحكم بالفأرة علي النافذة أثناء التشغيل.. ماذا إذا أردنا استخدام عجلة الفأرة Mouse Wheel علي إحدي أدوات التحكم .. وغيرها الكثير من الأمور التي نستطيع فعلها باستخدام طريقة خطفHook الرسائل..

التصنيف الفرعي للنوافذ Window Subclassing يجعلك قادر علي التحكم بالرسائل التي تصل لأي نافذة عن طريق خطف تلك الرسائل ومعالجتها قبل أن يتم إرسالها إلي تلك النافذة.

ومفهوم التصنيف الفرعي هو نفس مفهوم إجراء النافذة .. ما نقوم بعمله هو إلقاء خطاف لاصطياد رسائل نافذة ما وهنا نقوم بتغيير إجراء النافذة الافتراضي لإجراء فرعي ليكون هو إجراء النافذة .. وبداخل هذا الإجراء نقوم بتحليل الرسائل ومعالجتها ومن ثم الموافقة علي تمريرها إلي النافذة أم لا.

ولعمل ذلك نستخدم دالتين هما SetWindowLong , CallWindowProc .. وفيما يلي تفصيل لكلا منهما.

❖ دالة SetWindowLong :

تستخدم تلك الدالة لتغيير خصائص نافذة معينة .

The **SetWindowLong** function changes an attribute of the specified window. The function also sets a 32-bit (long) value at the specified offset into the extra window memory of a window.

```
LONG SetWindowLong(
  HWND hWnd,          // handle of window
  int nIndex,         // offset of value to set
  LONG dwNewLong      // new value
);
```

المعامل الأولhWnd يحدد مقبض تلك النافذة.

المعامل الثانيnIndex ويأخذ أي من القيم الثابتة التالية :

القيمة	المعني
GWL_EXSTYLE	تحديد سمات جديدة محسنة للنافذة مثل جعل النوافذ شفافة أو تغيير اتجاه الأدوات أو جعل النافذة تظهر ثابتة في مقدمة النوافذ وغيرها من السمات المحسنة التي تبدأ بـ[WS_EX_].
GWL_STYLE	تحديد سمات جديدة للنافذة.. مثل التحكم في زراير التكبير والتصغير والتحكم في قائمة النظام للنافذة..وكذلك حد النافذة .. وغيرها الكثير.
GWL_WNDPROC	**تحديد عنوان جديد لإجراء النافذة**
GWL_HINSTANCE	تحديد مقبض تطبيق جديد للنافذة.
GWL_ID	تحديد معرف جديد للنافذة

ما نريده لهذا المعامل هنا هو القيمة[GWL_WNDPROC] وتساوي[-4] .. تلك القيمة هي المسئولة عن تخصيص عنوان جديد لإجراء النافذة.

المعامل الثالث dwNewLong يحدد القيمة الجديدة .. وفي حالتنا هذه ستكون تلك القيمة هي عنوان إجراء النافذة الجديد.

وعند نجاح الدالة فهي تعود **بالقيمة السابقة** لإجراء النافذة الافتراضي.. لاحظ أننا نغيير عنوان إجراء النافذة الافتراضي إلي عنوان لإجراء جديد.. الدالة هنا تعود بعنوان إجراء النافذة الافتراضي .. ويفيد ذلك في شيئين .. **الأول** وهو إعادة توجيه الرسائل بعد معالجتها في التصنيف الفرعي لإجراء النافذة الجديد إلي إجراء النافذة الافتراضي .. **والثاني** وهو إذا أردنا فك الخطاف

وإعادة مسار الرسائل إلي إجراء النافذة الافتراضي مرة أخري بصورة مباشرة .. بدون وجود إجراء نافذة وسيط بين النظام والنافذة.

❖ **دالة CallWindowProc :**

تقوم الدالة بتمرير الرسائل إلي إجراء نافذة محدد.

The **CallWindowProc** function passes message information to the specified window procedure.

```
LRESULT CallWindowProc(
  WNDPROC lpPrevWndFunc,    // pointer to previous procedure
  HWND hWnd,                // handle to window
  UINT Msg,                 // message
  WPARAM wParam,            // first message parameter
  LPARAM lParam             // second message parameter
);
```

المعامل الأول lpPrevWndFunc يحدد مؤشر إلي إجراء النافذة السابق.. والذي حصلنا عليه من جراء القيمة المعادة من استدعاءSetWindowLong

المعامل الثاني hWnd وهو يحدد مقبض النافذة التي تم إرسال الرسالة لها.. والتي نريد أن تصلها تلك الرسالة.

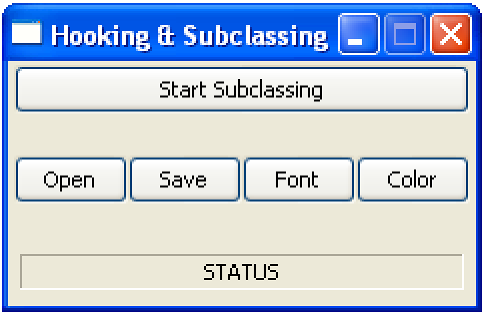
المعامل الثالث Msg وهو قيمة الرسالة.

المعاملان الرابع والخامس lParam , wParam وهي معلومات الرسالة الإضافية.

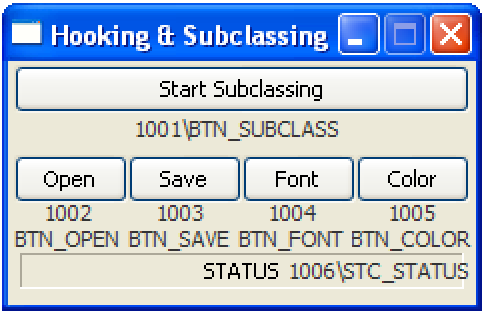
الآن سنتطرق لتصميم برنامج بسيط يحتوي علي المعلومات التي قمنا باستعراضها هنا.

أ – تصميم البرنامج

اسم المشروع الذي سنقوم بتصميمه هو Subclassing وسيحتوي علي خاصية تتيح عرض معلومات لحظية عن أداة الزر button لحظة مرور الماوس عليها..والشكل العام سيكون كالتالي:



والصورة التالية موضح عليها المعرفات واسماء الأدوات.



عند الضغط علي زر Start Subclassing سيتم إلقاء خطاف لاصطياد الرسائل التي تصل إلي الأزرار التالية Open,Save,Font,Color, وسيتحول الزر إلي Stop Subclassing .. وعند الضغط عليه مره أخري سيتم إيقاف اصطياد الرسائل.

وأثناء أصطياد الرسائل .. عند الحركة بمؤشر الفأرة علي أحد الأزرار الأربعة فسيتم عرض نبذه عن ما يفعله الزر في نافذة[STC_STATUS].
والآن نستعرض محتويات الملف Subclassing.Inc

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib

DlgProc            PROTO  :HWND,:UINT,:WPARAM,:LPARAM
BtnProc            PROTO  :HWND,:UINT,:WPARAM,:LPARAM
Hook               PROTO
UnHook             PROTO

.const
IDD_DIALOG1         equ 101
BTN_SUBCLASS        equ 1001
BTN_OPEN            equ 1002
BTN_SAVE            equ 1003
BTN_FONT            equ 1004
BTN_COLOR           equ 1005
STC_STATUS          equ 1006
MAX_STRING          equ 512
;###########################################
.data
ID_Btn_Start      db "Start Subclassing",0
ID_Btn_Stop       db "Stop Subclassing",0
Hook_Started      db "Subclassing Started.",0
Hook_Stopped      db "Subclassing Stopped.",0
```

```
32 Tip_Open          db "Click here to select and open a file",0
33 Tip_Save          db "Click here to save the file",0
34 Tip_Font          db "Click here to choose the font",0
35 Tip_Color         db "Click here to select the color",0
36
37 ;#########################################################
38 .data?
39 hInstance         dd ?
40 buffer            db 512 dup(?)
41 hOpen             dd ?
42 hSave             dd ?
43 hFont             dd ?
44 hColor            dd ?
45 hStatus           dd ?
46 lpPrevProc         dd ?
47 ;#########################################################
```

السطور 13-15 : تم الإعلان عن ثلاث إجرائات ..الأول وهو BtnProc إجراء التصنيف الفرعي للزراير **الثاني** Hook وخاص بإلقاء خطاف اصطياد الرسائل..**والثالث** UnHook وخاص بفك خطاف إصطياد الرسائل.

السطور 40-45 : متغيرات عددية ستحمل قيم مقابض نوافذ الأدوات.

السطر 46 : متغير عددي سيحمل قيمة عنوان إجراء النافذة الافتراضي السابق.

شفرة الملف *Subclassing.Asm*

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include Subclassing.inc

.code

start:

 invoke GetModuleHandle,NULL
 mov   hInstance,eax

 invoke InitCommonControls
 invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
 invoke ExitProcess,0

;########################################################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

 mov   eax,uMsg
 .if eax==WM_INITDIALOG
     invoke SetDlgItemText,hWin,BTN_SUBCLASS,addr ID_Btn_Start
     invoke GetDlgItem,hWin,BTN_OPEN
     mov hOpen,eax
     invoke GetDlgItem,hWin,BTN_SAVE
     mov hSave,eax
     invoke GetDlgItem,hWin,BTN_FONT
     mov hFont,eax
     invoke GetDlgItem,hWin,BTN_COLOR
     mov hColor,eax
     invoke GetDlgItem,hWin,STC_STATUS
     mov hStatus,eax
 .elseif eax==WM_COMMAND
     movzx eax,WORD PTR[wParam]
     movzx edx,WORD PTR[wParam+2]
     .if edx==BN_CLICKED
```

```
        .if eax==BTN_SUBCLASS
          invoke GetDlgItemText,hWin,BTN_SUBCLASS,addr buffer,MAX_STRING
          invoke lstrcmp,addr buffer,addr ID_Btn_Start
          jz StartSubclassing
          invoke SetDlgItemText,hWin,BTN_SUBCLASS,addr ID_Btn_Start
          invoke UnHook
          ret
          StartSubclassing:
           invoke SetDlgItemText,hWin,BTN_SUBCLASS,addr ID_Btn_Stop
           invoke Hook
        .endif

      .endif
  .elseif eax==WM_CLOSE
    invoke UnHook
    invoke EndDialog,hWin,0
  .else
    mov   eax,FALSE
    ret
  .endif
  mov   eax,TRUE
  ret

DlgProc endp

BtnProc  Proc hCtl:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
   .if uMsg == WM_MOUSEMOVE
      mov eax, hCtl
      .if eax == hOpen
         invoke SendMessage,hStatus,WM_SETTEXT,0,addr Tip_Open
      .elseif eax == hSave
         invoke SendMessage,hStatus,WM_SETTEXT,0,addr Tip_Save
      .elseif eax == hFont
         invoke SendMessage,hStatus,WM_SETTEXT,0,addr Tip_Font
      .elseif eax == hColor
         invoke SendMessage,hStatus,WM_SETTEXT,0,addr Tip_Color
      .endif
   .endif
  invoke CallWindowProc,lpPrevProc,hCtl,uMsg,wParam,lParam
  ret
BtnProc  endp
```

```
81 Hook Proc
82  invoke SetWindowLong,hOpen,GWL_WNDPROC,BtnProc
83  invoke SetWindowLong,hSave,GWL_WNDPROC,BtnProc
84  invoke SetWindowLong,hFont,GWL_WNDPROC,BtnProc
85  invoke SetWindowLong,hColor,GWL_WNDPROC,BtnProc
86  mov lpPrevProc,eax
87  invoke SendMessage,hStatus,WM_SETTEXT,0,addr Hook_Started
88  ret
89 Hook endp
90
91 UnHook Proc
92    invoke SetWindowLong,hOpen,GWL_WNDPROC,lpPrevProc
93    invoke SetWindowLong,hSave,GWL_WNDPROC,lpPrevProc
94    invoke SetWindowLong,hFont,GWL_WNDPROC,lpPrevProc
95    invoke SetWindowLong,hColor,GWL_WNDPROC,lpPrevProc
96    invoke SendMessage,hStatus,WM_SETTEXT,0,addr Hook_Stopped
97    ret
98 UnHook endp
99
100 end start
```

السطور 24-34 : نقوم بإرسال النص Start Subclassing إلي الزر ID_Btn_Start ومن ثم نقوم بإتيان مقابض نوافذ تحكم الأدوات.

السطر 36 : علي سبيل التغيير .. نقوم باستخدام التعليمة movzx والتي تستطيع أن تنسخ من مصدر بحجم أقل من حجم الهدف .. وهنا ننسخ قيمة معرف الأداة عند العنوان wParam بحجم كلمة Word أي 2 بايت ونضعها في مسجل بحجم كلمة مضاعفة DWord أي 4 بايت وهو eax.

السطر 37 : ننسخ قيمة الحدث عند العنوان wParam+2 بحجم 2 بايت Word ونضعها في المسجل edx.

السطور 39-49 : شفرة زر إلقاء الخطاف وفك الخطاف .. ونستطيع معرفة العملية المراد تنفيذها بطريقه مقارنة اسم الزر..ففي البداية نقوم بإتيان اسم الزر BTN_SUBCLASS .. ونقارنه بقيمة نص الاسم ID_Btn_Start فإذا كان ناتج المقارنة بصفر .. فمعني ذلك أن النص المعروض في الزر حاليا عند الضغط عليه هو [Start Subclassing] وبالتالي سيتم القفز إلي العنوان [StartSubclassing] وبه نستدعي الإجراء Hook لبدء تنفيذ إجراء اصطياد الرسائل وتغيير نص الزر إلي[Stop Subclassing].

بينما إذا كان ناتج المقارنة لا يساوي الصفر .. فهذا يعني أن خطاف الرسائل يعمل حاليا وأن النص في الز هو [Stop Subclassing] وسينتقل التنفيذ للسطر التالي وسيتم استدعاء إجراء فك خطاف الرسائل UnHook بالإضافة إلي تغيير نص الزر إلي[Start Subclassing].

ويمكن الوصول إلي نفس النتيجة بأكثر من طريقة .. وقد استخدمت هنا طريقة مقارنة نص الزر.

السطر 53 : من المفيد دائما أن نفك خطاف الرسائل دائما عند انهاء البرنامج .. في بعض المواقف تحدث أخطاء مع النظام .. إذا أنهينا البرنامج بدون فك الخطاف.

السطور 64-97 : شفرة الإجراء BtnProc وهي لا تختلف في شيء عن شفرة إجراء النافذة في البرنامج.. نقوم فيها بفحص الإشعار WM_MouseMove والذي تستقبله نافذة الزر عند حركة الماوس عليها.. ومن ثم نقوم بعمل مقارنة لمعرفة الزر الذي يستقبل هذا الإشعار .. ونقوم بإرسال المعلومة اللحظية الخاصة بالزر إلي نافذةSTC_STATUS.

وفي النهاية نستدعي دالة CallWindowProc لتمرير الرسالة إلي إجراء النافذة الافتراضي للأزار.

ويمكنك حجب استدعاء CallWindowProc إذا أردت أن لا تصل رسالة معينة إلي النافذة.

السطور 81-89 : شفرة إجراء إلقاء خطاف الرسائل.. نقوم فيه باستدعاء الدالة SetWindoLong وتمرير مقبض نافذة الأداة المراد مراقبة رسائلها والثابت GWL_WNDPROC .. وعنوان الإجراء BtnProc الذي سيستقبل تلك الرسائل.
ونقوم بتخزين قيمة عنوان إجراء النافذة الافتراضي في المتغيرlpPrevProc.

السطور 91-98 : شفرة إجراء فك خطاف الرسائل .. ونقوم باستدعاء دالة SetWindowLong مع تمرير عنوان إجراء النافذة الافتراضي لها في المعامل الأخير.



الفصل الحادي عشر

البحث عن الملفات

Finding Files

1. المفهوم وكيفية التطبيق.
2. إنشاء مسار تنفيذ Thread.
3. تصميم البرنامج.
4. شرح شفرة البرنامج

أ – المفهوم وكيفية التطبيق

باب البحث عن الملفات الموجودة علي الهارد ديسك من إحدي الموضوعات الهامة التي يجب الحديث فيها .. لإنه الأسلوب المتبع لتصميم مثل هذا البرنامج سيضيف إلي أفكارك الكثير الذي يمكن استغلاله بصور أخري.

وكبداية فإننا سنقوم بتصميم 4 إجراءات في هذا البرنامج ..

1 - إجراء خاص بإتيان الأقراص Fixed Drives الموجودة حاليا..
2 - إجراء خاص بإتيان المجلدات الموجودة ومجداتها الفرعية..
3 - إجراء خاص بإتيان الملفات الموجودة بكل مجلد..
4 - إجراء خاص بإنشاء مسار تنفيذ Thread سيلي الحديث عنه بتفصيل أكثر.

سنقوم حاليا باستعراض الدوال التي سنستخدمها في كل إجراء.

❖ دالة GetLogicalDrives :

تقوم تلك الدالة علي بإعادة قيمة بشكل قناع خانات ثنائية bitmask تمثل جميع الأقراص الموجودة حاليا.

The **GetLogicalDrives** function returns a bitmask representing the currently available disk drives.

DWORD GetLogicalDrives(VOID)

لا يوجد لتلك الدالة أي معاملات .. فقط يتم استدعائها وتعود بقيمة DWORD تمثل الأقراص الموجودة حاليا في نظام التشغيل .. في صورة قناع بتات ثنائي .. مكون من كل بت من الـ 32 بت في القيمة المعادة .. حيث إذا وجدت قيمة في بت معين .. فهنا نستطيع معرفة موضع هذا البت .. وبعد معرفة موضعه نستطيع معرفة القرص الذي يمثله.

وقناع البتات الثنائي موضح في الجدول التالي :

Drive	Bit Position	Binary	Decimal
A	0	1	1
B	1	10	2
C	2	100	4
D	3	1000	8
E	4	10000	16
F	5	100000	32
G	6	1000000	64
H	7	10000000	128
I	8	100000000	256
G	9	1000000000	512
K	10	10000000000	1024
L	11	100000000000	2048
M	12	1000000000000	4096
N	13	10000000000000	8192
O	14	100000000000000	16384

فمثلا فلنفرض أننا قمنا باستدعاء الدالة GetLogicalDrives وعادت بالقيمة العشرية 124 .. فهنا القيمة 124 نقوم بتمثيلها كالآتي :

124 = 1111100 = [4+8+16+32+64]

حيث عادت بـ 7 بتات ..
أول بت وهو صفر .. وهنا نعرف أنه لا يوجد قرصA.
ثاني بت وهو صفر .. وهنا نعرف أنه لا يوجد قرصB.
ثالث ورابع وخامس وسادس وسابع بتات قيمهم واحد .. وهنا نعرف أن الأقراص الموجودة هي [C,D,E,F,G].

ونقوم بمقارنة البتات مع قيم موضعها بالمعامل المنطقيAND.

فمثلا إذا أردنا معرفة هل القرصC موجود أم لا .. فنقوم بعمل الأتي:

124 AND 4

1111100 AND 100 = 100

بالتالي القيمة المعادة من مقارنة البت الثالث ليست بصفر وبالتالي نستطيع معرفة أن القرص موجود.. وإذا عادت بصفر فهذا يعني أن القرص ليس موجود.

مثال أخر إذا عادت الدالة بالقيمة 269 ..

269 = 100001101

بمجرد النظر هنا نستطيع معرفة الأقراص المتوفرة حاليا وهما [A,C,D,I].. قم بتطبيق المعامل المنطقي علي قيم الموضع .. وسيتم إثبات ذلك برمجيا.

❖ دالة GetDriveType :

دالة GetLogicalDrives كانت تعيد قيمة تمثل الأقراص الموجود حاليا .. ولكننا لم نعرف نوعية هذه الأقراص .. من الممكن أن يكون قرص مرن أو قرص صلب أو قرص الاسطوانات أو قرص ذاكرة مثل USB أو قرص شبكة.

وتتيح لك الدالة GetDriveType معرفة نوع القرص.

The **GetDriveType** function determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive.

```
UINT GetDriveType(
  LPCTSTR lpRootPathName        // address of root path
);
```

المعامل الوحيد لها lpRootPathName نقوم بتمرير مسار القرص الجذري علي الصيغة [C:] , [D:] وهكذا ..

وتعود الدالة بقيمة تحدد نوع القرص كما في الجدول التالي :

Value	Meaning
0	The drive type cannot be determined.
1	The root directory does not exist.
DRIVE_REMOVABLE	The drive can be removed from the drive.
DRIVE_FIXED	The disk cannot be removed from the drive.
DRIVE_REMOTE	The drive is a remote (network) drive.
DRIVE_CDROM	The drive is a CD-ROM drive.
DRIVE_RAMDISK	The drive is a RAM disk.

يمكننا هنا استثناء القيم الأخري والاكتفاء بالقيمة DRIVE_FIXED للبحث عن الملفات لأقراص الهارد الثابتة فقط.

وكان من الممكن الاستغناء عن دالة GetLogicalDrives وعمل تكرار لحروف الأقراص .. واختبار وجودها مع قيم مواضعها بالمعامل المنطقي AND.

❖ **دالة FindFirstFile :**

تقوم هذه الدالة بالبحث داخل مجلد عن الملف الأول الذي يطابق اسم الملف الممرر لها.

The **FindFirstFile** function searches a directory for a file whose name matches the specified filename. **FindFirstFile** examines subdirectory names as well as filenames.

```
HANDLE FindFirstFile(
  LPCTSTR lpFileName,               // pointer to name of file to search for
  LPWIN32_FIND_DATA lpFindFileData  // pointer to returned information
);
```

للدالة معاملين :

الأول lpFileName .. وهو مؤشر لنص يحوي اسم الملف المراد البحث عنه.. لاحظ هنا أننا يمكن استخدام الرمز الخاص (*) في البحث عن الملفات.

الثاني lpFindFileData وهو مؤشر لبنية لتحوي المعلومات المعادة من الدالة البنية هي بنية الملفات WIN32_FIND_DATA .. وتحتوي علي خصائص الملف الذي تم البحث عنها بواسطة دالة FindFirstFile أو دالة FindNextFile والتي سنتناولها بعد قليل.

The **WIN32_FIND_DATA** structure describes a file found by the FindFirstFile or FindNextFile function.

```
typedef struct _WIN32_FIND_DATA { // wfd
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD    nFileSizeHigh;
    DWORD    nFileSizeLow;
    DWORD    dwReserved0;
    DWORD    dwReserved1;
    TCHAR    cFileName[ MAX_PATH ];
    TCHAR    cAlternateFileName[ 14 ];
} WIN32_FIND_DATA;
```

أعضاء البنية كالتالي :

1 - dwFileAttributes : يحدد مجموعة من خصائص الملف تعود في صيغة قناع بتات ثنائية أيضا.. موضح بعضها في القيم التالية:

Value	**Bit**	**Binary**	**Decimal**
FILE_ATTRIBUTE_READONLY	0	1	1
FILE_ATTRIBUTE_HIDDEN	1	10	2
FILE_ATTRIBUTE_SYSTEM	2	100	4
FILE_ATTRIBUTE_DIRECTORY	4	10000	16
FILE_ATTRIBUTE_ARCHIVE	5	100000	32
FILE_ATTRIBUTE_ENCRYPTED	6	1000000	64
FILE_ATTRIBUTE_NORMAL	7	10000000	128
FILE_ATTRIBUTE_TEMPORARY	8	100000000	256
FILE_ATTRIBUTE_COMPRESSED	11	100000000000	2048

بالتالي إذا أردنا معرفة هل الملف الحالي الذي وجدته الدالة عبارة عن مجلد Directory .. فهنا نقوم باختبار قيمة dwFileAttributes بالمعامل المنطقي AND مع قيمة (10h) الممثلة في قيمة موضع البت.

2 - ftCreationTime : يحدد وقت إنشاء الملف.. تخزن في بنية FileTime.

3 - **ftLastAccessTime :** يحدد وقت أخر دخول للملف.

4 - **ftLastWriteTime :** يحدد وقت أخر كتابة للملف.

5 - **nFileSizeHigh :** يحدد قيمة 32 بت High Order .. تستخدم في حالة إذا كان حجم الملف كبير وتجاوز حجم الكلمة المضاعفة للعضو nFileSizeLow ..غير ذلك فالقيمة تكون صفر.

6 - **nFileSizeLow :** يحدد قيمة 32 بت Low Order تستخدم في تخزين حجم الملف.

7 - **dwReserved1-dwReserved0 :**
عضوان محجوزان للاستخدام المستقبلي.

8 - **cFileName :** يحمل اسم الملف.

9 - **cAlternateFileName :** يحمل اسم الملف بصيغة Fat16 .. 8 حروف لاسم الملف .. وثلاثة للامتداد.

وبعد نجاح دالة FindFirstFile فإنها تعود بقيمة موجبة لمقبض البحث .. ليستخدم بعد ذلك مع دالتي FindNextFile و FindClose.

وإذا فشلت الدالة فتعود بقيمة INVALID_HANDLE_VALUE قيمتها (-1) ..

الخلاصة أن دالة FindFirstFile تقوم بفتح مقبض للبحث وتعيد معلومات عن الملف الأول الذي وجدته تبعا لنمط الاسم الذي قمت بتمريره لها.

وكما ذكرنا يمكن استخدام الرمزين الشاملين (*) و (?) لتهيئة اسم البحث بشروط معينة.

❖ دالة FindNextFile :

تستخدم لاستكمال عملية البحث عن الملفات من استدعاء سابق لدالة FindFirstFile ..

The **FindNextFile** function continues a file search from a previous call to the FindFirstFile function.

```
BOOL FindNextFile(
  HANDLE hFindFile,                    // handle to search
  LPWIN32_FIND_DATA lpFindFileData  // pointer to structure
                                       for data on found file
);
```

وللدالة معاملين :

المعامل الأول **hFindFile** وهو مقبض البحث الذي حصلنا عليه من جراء استدعاء الدالة FindFirstFile.

المعامل الثاني **lpFindFileData** وهو لمؤشر للبنية WIN32_FIND_DATA.

وعند نجاح الدالة فهي تعود بقيمة غير الصفر .. وإذا فشلت فتعود بصفر.

وفي كل مرة يتم فيها استدعاء تلك الدالة فيتم إرجاع بيانات الملف التالي بالبنية WIN32_FIND_DATA.

❖ **دالة FindClose :**

تقوم هذه الدالة بإغلاق مقبض البحث بعد انتهائنا منه.

The **FindClose** function closes the specified search handle. The FindFirstFile and FindNextFile functions use the search handle to locate files with names that match a given name.

```
BOOL FindClose(
  HANDLE hFindFile        // file search handle
);
```

حيث المعامل الوحيد للدالة وهوhFindFile يحدد مقبض البحث المراد غلقة.

إلي هنا وصلنا نهاية الدوال المستخدمة في البحث عن الملفات .. الشفرة ستكون مقسمة إلي ثلاث إجراءات كما اتفقنا ..

أول إجراء وهو GetDrives خاص بإتيان الدريفات الموجودة علي نظام التشغيل..

بداخل هذا الإجراء .. يتم تمرير اسم الدريف .. لإجراء أخر وهوGetDirectories وهو خاص بإتيان مجلدات الدريف ..

وكل مجلد نقوم بإتيانه نقوم بتمرير مساره إلي إجراء أخر وهو GetFiles وهو يقوم بالبحث وإتيان الملفات **فقط** الموجودة بهذا المجلد .. وعرضها للمستخدم.

الإجراء GetDirectories سيكون إجراء رجعي Recursive .. بمعني أننا نقوم بإتيان الملفات .. فإذا كان بين هذه الملفات مجلد .. سنقوم بتمرير هذا المجلد إلي نفس الإجراء GetDirectories ليتم إتيان الملفات الموجودة به مرة أخري .. وهكذا.

2– إنشاء مسار تنفيذ Thread

إجراء البحث عن الملفات يتم داخل تكرار LOOP .. هذا التكرار يستهلك موارد النظام بصورة كبيرة .. كما يقوم هذا التكرار بتعطيل حلقة إجراء النافذة الحوارية DlgProc عن معالجة الرسائل التي تصل إليها ومنها يتوقف البرنامج عن الـ Responding لرسائل نظام التشغيل المرسلة إليها.

كل هذا يحدث لأن البرنامج يعمل علي مسار تنفيذ واحد Primary Thread .. بينما نستطيع أن ننشأ مسار تنفيذ جديد يقوم بمعالجة حلقة تكرار البحث عن الملفات .. ونترك مسار التنفيذ الافتراضي للبرنامج ليقوم بمعالجة رسائل النافذة.

ولعمل ذلك نستخدم الدالة التالية :

❖ دالة CreateThread :

تستخدم الدالة لإنشاء مسار تنفيذ جديد..

The **CreateThread** function creates a thread to execute within the address space of the calling process.

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  DWORD dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
```

وللدالة 6 معاملات موضحين كالآتي:

1 - lpThreadAttributes : مؤشر لبنية SECURITY_ATTRIBUTES تحدد الخصائص الأمنية لمسار التنفيذ.. ولن نستخدم هذا المعامل ونضع قيمته NULL.

2 - dwStackSize : يحدد الحجم الاستاك بالبايت الذي تريده لمسار التنفيذ الجديد .. وإذا تجاهلت هذا المعامل ووضعت قيمته NULL فسيتم حجز حجم مساوي لحجم مسار التنفيذ الافتراضي للتطبيق.

3 - lpStartAddress : معامل هام .. وهو يحدد عنوان بداية إجراء مسار التنفيذ الجديد.. ونضع له عنوان إجراء نقوم بإنشائه خاص لذلك.

4 - lpParameter : يستخدم في حالة إذا أردت تمرير معامل إلي إجراء المسار .. وإلا فنضع قيمته NULL .

5 - dwCreationFlags : معامل يحدد رايات إضافية لمسار التنفيذ تتحكم بعملية إنشائه.. ومن الممكن أن تكون الرايات مركبة من أكثر من راية حسب قناع البتات bitmask ..حيث البت الأول إذا كان بصفر فهذا يشير إلي أننا نريد استخدام المسار مباشرة بعد إنشائه .. بينما إذا كان البت الثالث بواحد فهذه راية CREATE_SUSPENDED .. وتشير إلي أننا نريد إنشاء مسار التنفيذ وإيقاف عمله لحينما نريد إعادة عمله بواسطة الدالة ResumeThread.

The **ResumeThread** function decrements a thread's suspend count. When the suspend count is decremented to zero, the execution of the thread is resumed.

```
DWORD ResumeThread(
  HANDLE hThread        // identifies thread to restart
 );
```

بينما يمكن إيقاف أي مسار تنفيذ يعمل حاليا بدالةSuspendThread.

The **SuspendThread** function suspends the specified thread.

```
DWORD SuspendThread(
  HANDLE hThread        // handle to the thread
 );
```

بينما إذا أردنا الخروج من مسار التنفيذ فيمكن استدعاءExitThread

The **ExitThread** function ends a thread.

```
VOID ExitThread(
  DWORD dwExitCode       // exit code for this thread
);
```

6 - lpThreadID : مؤشر لمتغير عددي 32 بت .. يستقبل معرف مسار التنفيذ..لن نحتاج لاستخدام هذا المعرف.

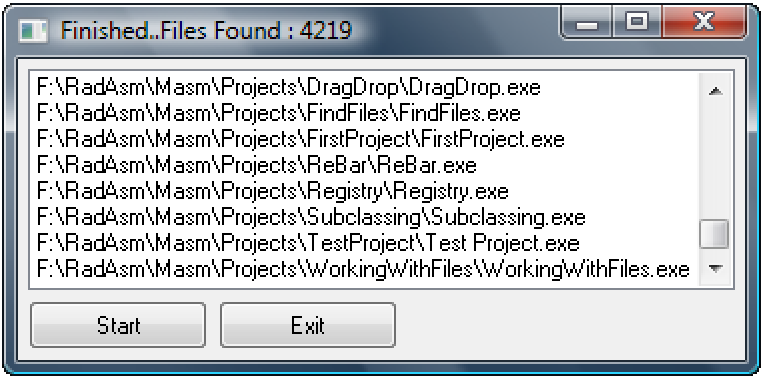
وبعد نجاح الدالة تعود بقيمة مقبض المسار .. وهو غير مهم ويمكننا غلقه بدالة CloseHandle ونمرر لها قيمة مقبض المسار .. وسيظل المسار يعمل في الخلفية.

```
BOOL CloseHandle(
  HANDLE hObject       // handle to object to close
);
```

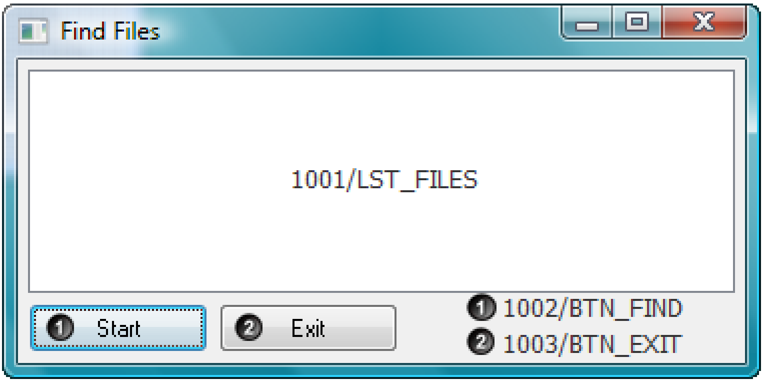
3- تصميم البرنامج

اسم المشروع الذي سنقوم بتصميمه هوFind Files .. وظيفته معروفة وهي البحث عن الملفات .. وعرضها في قائمة للمستخدم.

والشكل العام للنافذة سيكون كالتالي :



وأسماء الأدوات والمعرفات موضحة بالصورة التالية



تحتوي النافذة على قائمة لعرض مسارات الملفات بها .. وزر BTN_FIND يستخدم لبدء وإيقاف عملية البحث .. وزر BTN_EXIT يستخدم للخروج من البرنامج.

والآن نستعرض محتويات الملف[Find Files.Inc].

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc
include masm32.inc
include macros.asm

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib
includelib masm32.lib

DlgProc          PROTO  :HWND,:UINT,:WPARAM,:LPARAM
ThreadProc       PROTO
GetDrives        PROTO
GetDirectories   PROTO  :DWORD
GetFiles         PROTO  :DWORD

.const
IDD_DIALOG1          equ 101
LST_FILES            equ 1001
BTN_FIND             equ 1002
BTN_EXIT             equ 1003
MAX_STRING           equ 255
WM_ThreadEnd         equ WM_USER+100h
;########################################
```

```
.data
ID_APP_NAME         db "Find Files",0
ID_FIND_START       db "Start",0
ID_FIND_STOP        db "Stop",0
ID_EXIT             db "Exit",0
dHeight             dd 0
dWidth              dd 0
dTop                dd 0
dLeft               dd 0
Slash               db "\",0
SlashAll            db "\*",0
FileMask            db "\*.exe",0
Drives              dd 0
DriveLetter         db 2 DUP (0)
DriveColon          db ":",0
Drive               db 3 DUP(0)
CurDir              db ".",0
ParentDir           db "..",0
bNull               db NULL
buf                 db 512 dup(0)
dTMP1               dd 0
dTMP2               dd 0
bSTOP               BOOL TRUE
;########################################
.data?
hInstance           dd ?
hDlg                dd ?
hLst                dd ?
hFind               dd ?
hExit               dd ?
ThreadID            dd ?
;########################################
```

نقوم أولا بتضمين مكتبة الماكرو " macros.asm " لنستخدم منها الماكرو CTXT أو SADD المعدل بواسطة hutch .. والذي سيوفر علينا كتابة أسطر كود.

السطور 16-19 : نقوم بكتابة صيغة الإجراءات التي سنستخدمها.

السطر 27 : نقوم بإنشاء رسالة خاصة بنا .. لتستقبلها النافذة عندما ينتهي مسار التنفيذ .. ومعظم المبرمجين يستخدمون رسالة WM_USER كثابت ويضيفون لها قيمة في حدود 2^16 مع حذف 2^10 هي قيمة رسالة WM_USER .. المهم ألا تتساوي قيمة الرسالة مع رسالة أخري موجودة بالفعل.

السطر 39 : يحدد النمط المستخدم للبحث عن المجلدات.. حيث سنقوم بالبحث عن جميع المجلدات.

السطر 40 : يحدد النمط المستخدم للبحث عن الملفات .. وهنا استثنينا جميع الملفات ما عدا التي بها الامتداد "exe".

السطر 41 : متغير عددي سيحمل القيمة المعادة منGetLogicalDrives.

السطر 42 : متغير حرفي ليحمل رمز القرص.

السطر 43 : متغير حرفي ليحمل رمز الـcolon.

السطر 44 : متغير حرفي ليحمل رمز القرص مع الـcolon.

السطر 45 : متغير حرفي ليحمل رمز المجلد الحالي.

السطر 46 : متغير حرفي ليحمل رمز المجلد الأصلي.

السطور 49-50 : أثناء الانتقال بين الإجراءات .. بين الإجراء A والإجراء B فإن قيم المسجلات التي قمنا باستخدامها في الإجراء A ونعيد استخدامها في الإجراء B تتغير .. فبالتالي نحتاج لمتغيرات لتحمل قيم المسجلات داخل الإجراء A قبل الانتقال للإجراء B .. والتي نحتاج إليها عندما نعود للإجراء A مرة أخري.

السطر 51 : متغير سيحمل حالة البحث .. يعمل أم لا .. ونحدد عبره هل نريد تشغيل البحث أم إيقافه.

شفرة الملف Find Files.Asm

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive

include FindFiles.inc

.code

start:

 invoke GetModuleHandle,NULL
 mov   hInstance,eax

 invoke InitCommonControls
 invoke DialogBoxParam,hInstance,IDD_DIALOG1,NULL,addr DlgProc,NULL
 invoke ExitProcess,0

;###############################################

DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

 mov   eax,uMsg
 .if eax==WM_INITDIALOG
    push hWin
    pop hDlg
      invoke GetDlgItem,hWin,LST_FILES
      mov hLst,eax
      invoke GetDlgItem,hWin,BTN_FIND
      mov hFind,eax
      invoke GetDlgItem,hWin,BTN_EXIT
      mov hExit,eax

      invoke SetDlgItemText,hWin,BTN_FIND,addr ID_FIND_START
      invoke SetDlgItemText,hWin,BTN_EXIT,addr ID_EXIT

 .elseif eax==WM_COMMAND
      movzx eax,word ptr[wParam]
      movzx edx,word ptr[wParam+2]
      .if edx==BN_CLICKED
         .if eax==BTN_FIND
```

```
          invoke GetDlgItemText,hWin,BTN_FIND,addr buf,MAX_STRING
          invoke lstrcmp,addr buf,addr ID_FIND_START
          jz StartFinding
          mov bSTOP,TRUE
          invoke SetDlgItemText,hWin,BTN_FIND,addr ID_FIND_START
          ret
        StartFinding:
          invoke SetDlgItemText,hWin,BTN_FIND,addr ID_FIND_STOP
          mov bSTOP,FALSE
          invoke SendMessage,hLst,LB_RESETCONTENT,0,0
          mov eax,offset ThreadProc
          invoke CreateThread,NULL,NULL,eax,NULL,0,ADDR ThreadID
          invoke CloseHandle,eax

      .elseif eax==BTN_EXIT
        invoke SendMessage,hWin,WM_CLOSE,0,0
      .endif
    .endif

.ELSEIF uMsg==WM_ThreadEnd
    mov bSTOP,TRUE
    invoke SetDlgItemText,hWin,BTN_FIND,addr ID_FIND_START
    invoke SendMessage,hLst,LB_GETCOUNT,0,0
    mov  ebx,eax
    invoke wsprintf, addr buf, CTXT("Finished..Files Found : %d"), ebx
    invoke SendMessage,hWin,WM_SETTEXT,0,addr buf

 .elseif eax==WM_SIZE
    movzx edx,word ptr [lParam]    ;Client Height
    movzx eax,word ptr [lParam+2]  ;Client Width
    mov dHeight,eax
    mov dWidth,edx
    mov dLeft,5
    mov dTop,5
    sub dHeight,40
    sub dWidth,10
    invoke MoveWindow,hLst,dLeft,dTop,dWidth,dHeight,TRUE
    mov dWidth,90
    mov eax,dHeight
    add eax,10
    mov dTop,eax
    mov dWidth,90
    mov dHeight,25
```

```
      invoke MoveWindow,hFind,dLeft,dTop,dWidth,dHeight,TRUE
      mov dLeft,100
      invoke MoveWindow,hExit,dLeft,dTop,dWidth,dHeight,TRUE

 .elseif eax==WM_CLOSE
  invoke EndDialog,hWin,0
 .else
  mov   eax,FALSE
  ret
 .endif
 mov   eax,TRUE
 ret

DlgProc endp

;##############################################

ThreadProc PROC
 call GetDrives
 invoke PostMessage,hDlg,WM_ThreadEnd,NULL,NULL
 invoke ExitThread,0
 ret
ThreadProc ENDP

;##############################################

GetDrives Proc
 invoke GetLogicalDrives
 mov Drives,eax
 mov ebx,1
 mov esi,0
  .Repeat
   .if bSTOP
    .break
   .endif
   mov edx,Drives
   mov edi,65
   and edx,ebx
   jz NextDrive
   add edi,esi
   mov eax,edi
   mov BYTE ptr [DriveLetter], al
   invoke lstrcpy,addr Drive,addr DriveLetter
```

```
  invoke lstrcat,addr Drive,addr DriveColon
  invoke GetDriveType,addr Drive

  mov dTMP1,ebx
  mov dTMP2,esi

  .if eax==3
      invoke GetDirectories, addr Drive
  .endif
  mov ebx,dTMP1
  mov esi,dTMP2

 NextDrive:
  add ebx,ebx ;Position
  inc esi
.Until esi==26
 ret
GetDrives endp

;###############################################

GetDirectories Proc dDir:DWORD
  LOCAL DirPath[MAX_PATH]:BYTE
  LOCAL Struct_Dir:WIN32_FIND_DATA
  LOCAL hDir:DWORD

 invoke GetFiles,dDir

 invoke RtlZeroMemory, addr DirPath, sizeof DirPath
 invoke lstrcpy, addr DirPath, dDir
 invoke lstrcat, addr DirPath, addr SlashAll

 invoke FindFirstFile, addr DirPath, addr Struct_Dir
 .if (eax != INVALID_HANDLE_VALUE)
  mov hDir, eax
  .repeat
    .if bSTOP
      .break
    .endif
    invoke lstrcmp, addr Struct_Dir.cFileName,addr CurDir
    mov ebx, eax
    invoke lstrcmp, addr Struct_Dir.cFileName,addr ParentDir
    .if  (eax && ebx)
```

```
            mov eax, Struct_Dir.dwFileAttributes
            and eax, FILE_ATTRIBUTE_DIRECTORY
           .if (eax)
               invoke lstrcpy,addr DirPath,dDir
               invoke lstrcat,addr DirPath,addr Slash
               invoke lstrcat,addr DirPath,addr Struct_Dir.cFileName
               invoke GetDirectories, addr DirPath
            .endif
       .endif
       invoke FindNextFile, hDir, addr Struct_Dir
    .until (!eax)
     invoke FindClose, hDir
  .endif

  ret
GetDirectories endp

;###############################################

GetFiles Proc dDir:DWORD
    LOCAL buffer[512]:BYTE
    LOCAL DirPath [MAX_PATH]:BYTE
    LOCAL Struct_File:WIN32_FIND_DATA
    LOCAL hDir:DWORD

  invoke RtlZeroMemory, addr DirPath, sizeof DirPath
  invoke lstrcpy, addr DirPath, dDir
  invoke lstrcat, addr DirPath, addr FileMask

  invoke FindFirstFile, addr DirPath, addr Struct_File
  .if (eax != INVALID_HANDLE_VALUE)
    mov hDir, eax
    .repeat
       .if bSTOP
          .break
       .endif
       invoke lstrcmp, addr Struct_File.cFileName,addr CurDir
       mov ebx, eax
       invoke lstrcmp, addr Struct_File.cFileName,addr ParentDir
       .if (eax && ebx)
            mov eax, Struct_File.dwFileAttributes
            and eax, FILE_ATTRIBUTE_DIRECTORY
           .if (!eax)
```

```
                invoke lstrcpy,addr buffer,dDir
                invoke lstrcat,addr buffer,addr Slash
                invoke lstrcat,addr buffer,addr Struct_File.cFileName
                invoke SendMessage,hLst,LB_INSERTSTRING,-1,addr buffer
                invoke RtlZeroMemory, addr buffer, sizeof buffer
                invoke SendMessage,hLst,LB_GETCOUNT,0,0
                mov  ebx,eax
                invoke wsprintf, addr buffer, CTXT("Files Retreived [%d]"), ebx
                invoke SendMessage,hDlg,WM_SETTEXT,0,addr buffer
               ;invoke SendMessage,hLst,WM_PAINT,0,0
               ;invoke SendMessage,hDlg,WM_PAINT,0,0
           .endif
         .endif
         invoke FindNextFile, hDir, addr Struct_File
    .until  (!eax)
     invoke FindClose, hDir
  .endif

  ret
GetFiles endp

end start
```

4- شرح شفرة البرنامج

السطور24-31 : نقوم بتخزين قيم مقابض الأدوات والنافذة الحوارية.

السطور 33-34 : نقوم نقوم بتسمية الزرين Start,Exit.

السطور 40 : شفرة زر بدء وإيقاف البحث.

السطر 41 : نقوم بمعرفة النص الموجود بالزر BTN_FIND.

السطر 42 : نقوم بمقارنة نص الزر بنص البدء .. ومن الأفضل هنا أن نستخدم للمقارنة المتغير bSTOP .. أفضل كثيرا من مقارنة النص.

السطر43 : إذا كان النصين متساويان فهذا يعني أن الزر به النص Start ونحتاج لبدء تنفيذ البحث وبالتالي يتم القفز إلي StartFinding .

السطر 44 : إذا كان النصين غير متساويان فهذا يعني أن الزر به النص Stop .. ونحتاج لإيقاف البحث ..وبالتالي نمرر القيمة True لمتغير الحالة bSTOP.

السطر 45: ونرسل نص Start إلي الزر .. وهي خطوة زائده لا حاجة لوضعها هنا.. لإنه سيلي وضعها في مكانها الصحيح بالسطر62.

السطر 48 : نقوم بتغيير نص الزر إلي Stop.

السطر 49 : وننقل القيمة False للمتغير bSTOP لنحدد أن عملية البحث قد بدأت.

السطر 50 : نرسل رسالة حذف محتويات القائمة LB_RESETCONTENT.

السطر 51 : نقوم بالحصول علي عنوان الإجراء ThreadProc.

السطر 52 : نقوم باستدعاء دالة CreateThread لإنشاء مسار التنفيذ.. ونمرر لها عنوان مسار التنفيذ وكذلك عنوان معرف المسار والذي لن نستخدمه.

السطر 53 : نقوم بتحرير مقبض المسار .. ولن يؤدي ذلك إلي غلقه.

السطر 60 : عند إنتهاء عمل مسار التنفيذ فسيتم إرسال الإشعار WM_ThreadEnd إلي نافذة الصندوق الحواري بناء علي طلبنا.

السطر 61 : نتأكد أن قيمة bSTOP بـ TRUE ليعلن أن حالة البحث متوقفة.

السطر 62 : نقوم بتغيير نص الزر إليStart.

السطر 63 : نقوم بالحصول علي عدد العناصر الموجودة بالقائمة بإرسال رسالة LB_GETCOUNT.

السطر 64 : نقوم بنسخ عدد العناصر المعادة إلي المسجلebx.

السطر 65 : نقوم باستخدام الماكرو CTXT والذي يريحنا من عناء كتابة كود لحجز متغير حرفي وتهيئته لنص معين .. حيث نقوم باستدعاء الماكرو ... ويقوم هو بحجز مساحة علي الاستاك يخزن بها النص ويعود بعنوانه.. من المريح دائما أن تستخدم هذا الماكرو مع النصوص الديناميكية التي لا تحتاج إليها سوي للإجراء الحالي.

أيضا قمنا باستدعاء دالة wsprintf .. إحدي الدوال الشهيرة في عملية تنسيق وتهيئة النصوص.

The **wsprintf** function formats and stores a series of characters and values in a buffer. Any arguments are converted and copied to the output buffer according to the corresponding format specification in the format string. The function appends a terminating null character to the characters it writes, but the return value does not include the terminating null character in its character count.

```
int wsprintf(
  LPTSTR lpOut,       // pointer to buffer for output
  LPCTSTR lpFmt,      // pointer to format-control string
  ...                 // optional arguments
);
```

حيث :

المعامل الأول lpOut :

نمرر به عنوان المخزن buffer الذي سيتم وضع نتيجة النص به.

المعامل الثانيlpFmt :

نمرر به عنوان النص المراد تهيئته .. يتخلل النص رموز الإقحام التالية والتي يجب أن يسبقها علامة[%] :

الرمز	المعني
C	إقحام حرف واحد Character
d	إقحام قيمة عشرية Decimal
S	إقحام نص String
#	إضافة البادئة 0x للقيم السداسي عشرية

المعامل الثالث[...] :

عبارة عن معاملات إضافية .. تبعا لعدد رموز الإقحام التي ألحقتها بالمعامل الثاني.. ونقوم بتمرير القيم المراد إقحامها في النص هنا.

الخلاصة أنه سيتم إقحام قيمة المسجلebx داخل النص :
"Finished..Files Found : %d"

بدلا من الرمز %d.. وسيتم نسخ النص النهائي إلي المخزنbuf.

السطر 66 : نقوم بإرسال رسالة WM_SETTEXT لتغيير نص عنوان النافذة إلي نص buf .. لعرض نتيجة عدد الملفات التي تم إتيانها.

السطور 68-86 : ضبط أحجام وموضع الأدوات تبعا للتغيير في حجم النافذة .. كما استعرضنا من قبل.

السطور 101-106 : شفرة إجراء مسار التنفيذ ThreadProc .. حيث نقوم بها بإستدعاء الإجراء GetDrives .. وبعد انتهاء إجراء GetDrives سيتم العودة إلي الإجراء ThreadProc مرة أخري لنقوم بإرسال الإشعار WM_ThreadEnd إلي النافذة.

السطر 110 : إجراء إتيان الأقراص الموجودة علي النظام.

السطر 111 : نقوم بإستدعاء الدالة GetLogicalDrives.

السطر 112 : نقوم بنسخ القيمة المعادة والتي تمثل الأقراص الموجودة في قناع ثنائي إلي المتغير العدديDrives.

السطر 113 : المسجل ebx هنا سيحمل قيم مواضع البتات 1-2-4-8-16-32 وهكذا سيمثل 26 قيمة بحروف الأقراص.. ونبدأها بالقيمة [1] وهي قيمة البت رقم صفر.

السطر 114 : المسجل esi هنا سيكون عداد التكرار من الصفر إلي 25 .. وهو الذي سيعرفنا اسم حرف القرص إذا كانت قيمة موضعه غير الصفر.

السطر 115 : بدء حلقة تكرارية .

السطور 116 – 118 : من المفيد داخل الحلقة أن نعرف علي هل المستخدم طلب إيقاف البحث .. وبالتالي نقوم بالخروج من التكرار.

السطر 119 : نقوم بنسخ قيمة الأقراص الموجودة للمسجلedx.

السطر 120 : لاحظ أن حرف A يقابله في جدول الآسكي القيمة 65 .. وحرف B القيمة 66 .. وحرف C القيمة 67 .. وهكذا .. وهنا نقوم بتهيئة المسجل edi ليحمل القيمة 65 في كل لفة .. ونضيف عليه قيمة عداد التكرار esi إذا كان القرص موجودا .. فمثلا إذا كنا في الدورة رقم 3 وكان القرص موجود في هذه الدورة .. فنضيف [65+2] لتكون 67 .. ويكون لدينا هنا قيمة الحرف [C].

السطر 121 : نقوم بتطبيق المعامل المنطقي بين قيمة موضع البت الحالي وبين قيمة الأقراص المعادة.

السطر 122 : ونختبر هنا إذا كان الناتج القيمة صفر فهذا يعني أن القرص الحالي غير موجود ونقفز لـNextDrive لإختبار وجود قرص أخر.

السطر 123 : إلي هنا يكون القرص موجود بالفعل .. وبالتالي نقوم كما ذكرنا بإضافة عداد التكرار إلي القيمة65 لمعرفة قيمة حرف الآسكي للقرص.

السطر 124 : نقوم بنسخ رقم الآسكي من المسجلedi إلي eax.

السطر 125 : وتوجد أكثر من طريقة لتحويل رقم الحرف الآسكي إلي حرف آسكي .. من الممكن استخدام دالة wsprintf والإقحام بواسطة "%d" .. أو نقوم باستخدام دالة dwtoa .. أو نستخدم الطريقة التقليدية بنسخ قيمة المسجل الذي تم وضع رقم الآسكي به [al] إلي أول بايت في عنوان المتغير النصي DriveLetter.

السطر 126 : نقوم بنسخ حرف القرص إلي المتغير النصيDrive.

السطر 127 : نقوم بإلحاق رمز الـ colon إلي المتغير النصي Drive.

السطر 128 : نقوم باستدعاء الدالة GetDriveType وتمرير المتغير النصي Drive إليها لمعرفة نوع القرص.

السطر 130 : نقوم بتخزين مؤقت لقيمة المسجل ebx قبل الانتقال من الإجراء الحالي إلي إجراء جديد.

السطر 131 : نقوم بتخزين مؤقت لقيمة المسجل esi قبل الانتقال من الإجراء الحالي إلي إجراء جديد.

السطور 133-135 : نختبر هل القرص Fixed_Drive .. فإذا كان فنقوم باستدعاء الإجراء GetDirectories وتمرير عنوان القرص له.

السطر 136 : نعيد قيمة المسجل ebx مرة أخري إليه.

السطر 137 : نعيد قيمة المسجل esi مرة أخري إليه.

السطر 140 : طريقة لحساب قيمة موضع البت bit position .. وهي أن نقوم بجمع قيمة موضع البت علي نفسه في كل دورة.

السطر 141 : وبالطبع نزيد عداد التكرار esi في كل دورة إلي أن يصل إلي قيمته إلي 26 فينتهي التكرار.

السطور 148- 151 : شفرة إجراء GetDirectories .. ولاحظ أن به معامل dDir يستقبل مؤشر لعنوان المسار الذي سيبحث به .. ونقوم بحجز البنية Struct_Dir والتي ستحمل بيانات الملفات التي يتم البحث عنها.. كذلك DirPath سيحمل صيغة مسار البحث وhDir سيحمل مقبض البحث.

السطر 153 : نقوم باستدعاء الإجراء GetFiles لإتيان الملفات الموجودة بالمسار الممرر للإجراء الحالي.

السطر 155 : نستدعي دالة RtlZeroMemory والتي تقوم بملء جزء من الذاكرة بقيم صفرية.. ونستخدمها لتفريغ المتغيرDirPath من أي بيانات به.

```
VOID RtlZeroMemory(
    IN VOID UNALIGNED  *Destination,
    IN SIZE_T  Length
    );
```

المعامل الأول وهو عنوان الذاكرة والتي ستبدأ عملية الملء منه.
المعامل الثاني ويحدد عدد البايتات المراد ملئها.

السطر 156 : نقوم بنسخ مسار العنوان الممرر للإجراء إلي المتغير الداخلي DirPath.

السطر 157 : نقوم بإلحاق "*\" إلي نهاية المتغير DirPath لتكون صيغة البحث عن المجلدات.

السطر 159 : نستدعي دالة FindFirstFile ونمرر لها صيغة البحث .. بالإضافة إلي البنية التي ستحمل بيانات الملف.

السطر 160 : نختبر القيمة المعادة من الدالة إذا لم تكن قيمتها [1-] أو INVALID_HANDLE_VALUE فنقوم بالاستمرار.

السطر 160 : ننسخ مقبض البحث لدينا فيhDir.

السكر 162 : نقوم بفتح حلقة تكرارية لإتيان جميع الملفات طالما أن القيمة المعادة من جراء استخدامFindNextFile غير الصفر.

السطور163-165 : أيضا نختبر هل طلب المستخدم إيقاف البحث أم لا.

السطور166- 169 : من ضمن الملفات التي تعود خلال البحث المجلدات الحالية والمجلدات الجذرية..هذه المجلدات ليست مجلدات حقيقية بل هي روابط وصل لمجلدات .. وإذا لم نضعها في الاعتبار فسيدخل التكرار في حلقة غير منتهية .. فمثلا تخيل أننا في مجلد الويندوز " C:\Windows" وخلال البحث يعود لدينا المجلد الحالي "." ونمرره للإجراء مرة أخري ليبحث بداخله .. وهنا نقوم بتمرير نفس مسار المجلد "C:\Windows" .. وهكذا ندخل في حلقة غير منتهية .. ولهذا نحتاج لمعرفة إذا كان الملف الحالي ليس أي من المجلدين عن طريق:

1 - مقارنة اسم الملف باسم المجلد الحالي[.]
2 - تخزين ناتج المقارنة في المسجلebx
3 - مقارنة اسم الملف باسم المجلد الأصلي[..]
4 - تطبيق المعامل المنطقيAnd علي كل من النتيجتين

في الخطوة 1 إذا لم يكن الملف هو المجلد الحالي [.] فستعود الدالة بالقيمة True والعكس.

في الخطوة 3 إذا لم يكن الملف هو المجلد الأصلي [..] فستعود الدالة بالقيمة True والعكس ..

وعند تطبيق المعامل المنطقي
TRUE AND TRUE = TRUE

وبالتالي سيتحقق الشرط فقط إذا لم يكن الملف أي من المجلدين .. وإذا كان أي من المجلدين .. فلن يتحقق الشرط وسيعود بـ False وننتقل للبحث عن الملف التالي مباشرة..ونتجاهل هذا المجلد.

السطر 170 : نقوم بنسخ خصائص الملف إلي المسجل eax.

السطر 171 : نقوم بتطبيق المعامل المنطقي AND لمعرفة هل الملف الحالي عبارة عن مجلد أم لا.

السطور 173-177 : فإذا كان مجلد فسيعود بنتيجة غير الصفر .. وبالتالي سنحتاج هنا إلي تجهيز مسار المجلد .. وإرساله إلي الإجراء GetDirectories مرة أخري لإتيان الملفات الموجودة به.

حيث نقوم أولا بنسخ مسار المجلد للمتغير DirPath .. ثم نضيف الشرطة المائلة "\" ثم نضيف اسم المجلد الجديد الذي حصلنا عليه .. ونمرر عنوان المجلد الفرعي الجديد DirPath للإجراء مرة أخري.

السطر 179 : نقوم باستدعاء دالة FindNextFile وتمرير مقبض البحث لها .. وكذلك عنوان البنية التي ستستقبل بيانات الملف.

السطر 180 : نختبر القيمة المعاداة من الدالة FindNextFile فإذا عادت بقيمة غير صفرية فسيتحقق شرط الاستمرار (eax!) والذي يستمر طالما لم نحصل علي قيمة غير صفرية [Until Zero] وهذا يعني أن الدالة نجحت وتم الحصول علي ملف جديد وبالتالي نعود لبداية التكرار مرة أخري.

بينما إذا عادت الدالة بقيمة صفرية .. فسيكون الخروج من التكرار هنا متحقق [Until Zero] .

السطور 189 – 232 : شفرة إتيان الملف .. وهي تشبه تماما شفرة إتيان المجلدات ولكننا هنا نختبر الملف بواسطةFILE_ATTRIBUTE_DIRECTORY ومن ثم نقوم بتجهيز مساره وإضافته في القائمة .. وإخبار المستخدم أثناء البحث بعدد الملفات التي وجدناها .. يمكنك عمل متغير عداد لعدد الملفات أو الاعتماد علي عدد عناصر القائمة .

ومن الممكن أن تريد إعادة رسم النافذة أو القائمة بواسطة رسالة WM_PAINT إذا وجدت مشكلة في عرض شكل النافذة أثناء العمل.

فهرس المحتويات